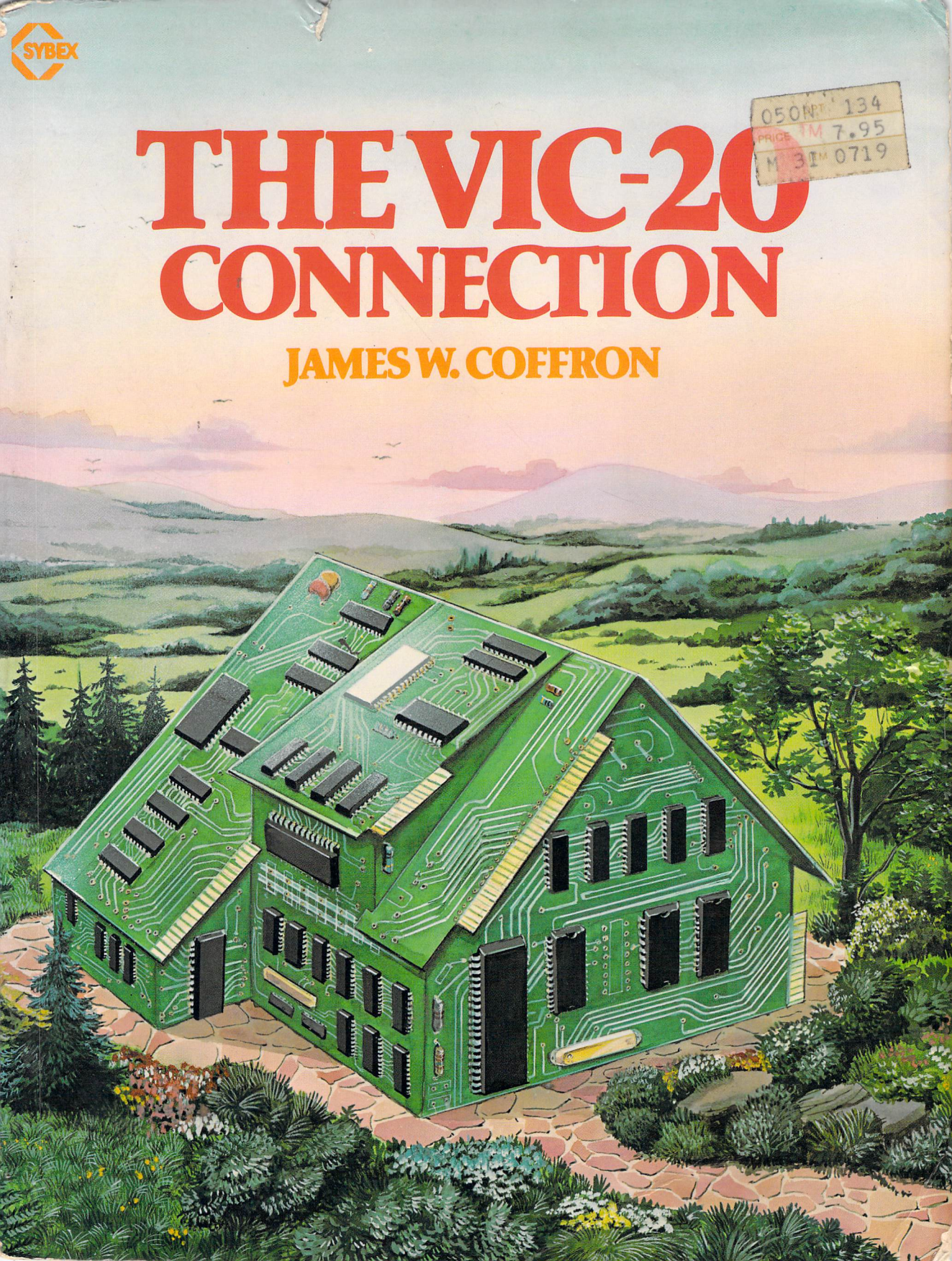
SYBEX



# THE VIC-20 CONNECTION

JAMES W. COFFRON



# THE VIC-20 CONNECTION

# THE VIC-20™ CONNECTION

JAMES W. COFFRON


Berkeley • Paris • Düsseldorf

Cover design by Daniel Le Noury
Book layout and design by Sharon Leong
Technical Illustration by Sharon Leong



Library of Congress Card Number: 83-50227
ISBN 0-89588-128-4
First Edition 1983
Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

*for the Spiesmans: John, Mary, Bill and Anne.*

# ACKNOWLEDGEMENTS

I wish to thank my wife Carol, for preparing the original diagrams; and Jim Compton, for editing the manuscript. Finally, a hearty thanks to the entire staff at Sybex for their splendid efforts.

# CONTENTS

# PREFACE

If you have recently purchased a VIC-20 Personal Computer™, or are thinking of purchasing one, many questions about the system have probably arisen in your mind. Many new computer owners are curious about the potential extent of their computer's overall usefulness. While it is true that a specific use for the system is probably the reason you bought it in the first place, you may wonder what else it can do.

It is safe to say that there are more potential future uses of the system than any purchaser can dream of at present. As you sit in front of the computer, you can look forward to many hours of enjoyment using the numerous available applications programs, and the various "off-the-shelf" games that can be played on the system. If you are a beginner in home computers, these games and programs may seem difficult to master at first, but this difficulty will soon pass, so do not fear.

At first you may be hesitant to use the system. A "fear of the unknown" surfaces as you test the machine's reactions to your nimble (or not-so-nimble) touches on the keys. Then, your boldness and confidence improve as you discover it is OK to make mistakes. Nothing drastic happens when you press the wrong key. The VIC-20 computer is a very forgiving instrument.

Before long, you are deftly running, modifying, and writing programs. The once-formidable task of using a home computer has diminished. Soon you find yourself looking for new challenges and new applications for the computer. At this point you may start to wonder, "Can I use the computer around the home? Is it possible to control my appliances, heating, or security systems with my computer?"

You know these things are possibilities, because you have read about them. However, you may feel it is far beyond your ability to accomplish them. If you think that, you will soon see that you are wrong. The realization of these controls with your VIC-20 is not beyond your capabilities. The information required may be different from that which you use everyday, but making *the VIC-20 Connection* is a straightforward process, and not very complex.

The designers of the VIC-20 computer have used valuable foresight in anticipating that system users who do not know or care much about hardware may want to create new interfaces between their system and the outside world. With that in mind, the VIC-20 was designed to make the interfacing job an easy one. You do not have to be a computer expert to construct the hardware for controlling external devices or to write the software for control. You will see how easy it is as you progress through the examples outlined in the text.

The low cost and flexibility of the VIC-20 make this machine especially suitable for applications that require a computer to be "dedicated" to a single task and left in place, monitoring temperature, for example. Several VIC-20 computers can be bought for the price of one more expensive machine, and at least one can be saved for routine computing tasks.

So if you are ready to enter the world of computer control, this book is the first step. It will open the door and provide you with the essential information needed to connect your computer to a variety of peripheral devices.

Without any further hesitation, turn the pages and learn to make *the VIC-20 Connection.*

# INTRODUCTION

This book is written for everyone who wants to understand how the VIC-20 Personal Computer, as well as other home computers, can be interfaced to the outside world. Specific examples are shown, using the VIC-20 to illustrate the essential concepts of computer control and interfacing. However, the information and ideas presented here can be readily adapted to most home computer systems.

Interfacing and controlling external devices with the VIC-20 will involve the use of both software and hardware. To that end, this text assumes the reader can write simple programs in VIC BASIC; an extensive knowledge of BASIC is not required to get the maximum value from this text. The hardware concepts are presented with the understanding that many readers may not be familiar with digital electronics. You do not have to be a software or hardware expert to make good use of the information given in this text.

This book is organized so as to enable the reader to understand how all the pieces of the interfacing and control puzzle fit together. The path is not complicated, but you will have to learn some new information and concepts. All the essential information for interfacing and controlling external devices with the VIC-20 PC is given in the pages that follow.

**Chapter 1** starts off with an introduction to, and a definition of, the concept of computer control, and presents some new vocabulary.

**Chapter 2** discusses the software required to output information from the VIC-20 computer to an external device. The programming language used is BASIC.

**Chapter 3** discusses the software required to input information into the VIC-20 computer from an external device. Again, BASIC is used as the programming language.

**Chapter 4** introduces the basic hardware concepts necessary to input and output information to and from the VIC-20 computer. This chapter is designed for readers who are unfamiliar with digital electronics, and want to learn only as much about it as they need for practical purposes.

**Chapter 5** presents an application of computer control in the design of a home security system. It starts with a definition of the problem and works through the software and hardware concepts necessary to have a working home security system. After reading this chapter, you will have a good general idea of how to use the computer in a home security application.

**Chapter 6** shows how to make the VIC-20 computer produce speech. The discussion starts with an introduction to speech synthesis using phoneme speech techniques, and then goes over the software and hardware required to let your VIC-20 speak, saying whatever you want, whenever you want.

**Chapter 7** discusses the difference between the concepts *analog* and *digital,* using examples comparing how various real-world events each would appear in an analog or digital environment. This chapter concludes with an explanation of the term *transducer.*

**Chapter 8** shows how to perform analog-to-digital conversion with the computer. General concepts applicable to most home computers are discussed. An actual analog-to-digital converter is connected to the VIC-20 computer, with all important software and hardware details shown. The chapter concludes with a complete hardware and software system for measuring temperature with the VIC-20.

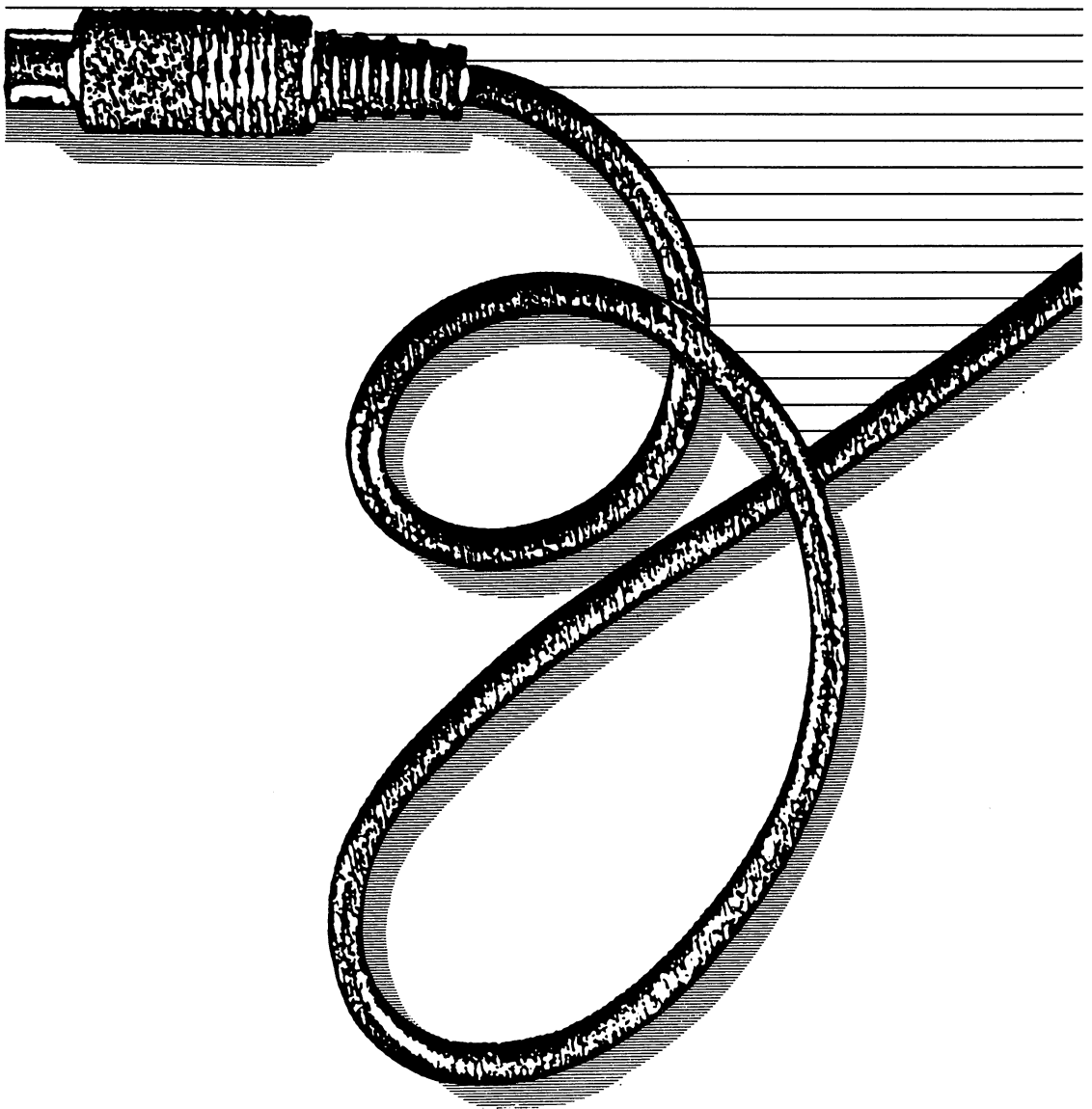
**Chapter 9** presents the opposite of analog-to-digital conversion, digital-to-analog conversion. The basic concepts are introduced, and

an actual digital-to-analog converter is connected to the computer. All of the important aspects of the software and hardware for digital-to-analog conversion are covered.

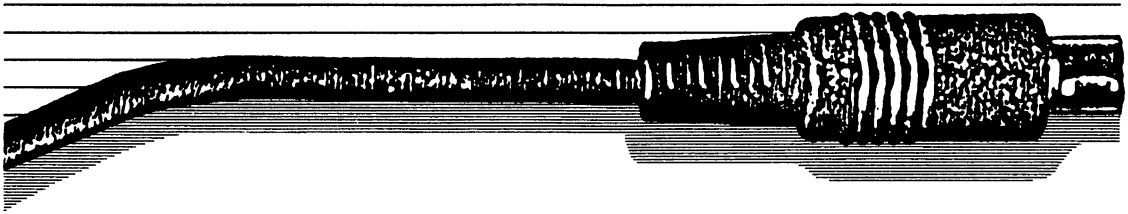
Finally, the **Appendices** present some useful reference information: a glossary, instructions for reading schematic diagrams, manufacturers' data sheets, and a list of vendors for the equipment described in this book.

The use of computers to control and monitor the environment around the home is increasing and will continue to increase in the future. If you want to become involved in the exciting field of computer control, this book is a good starting point.

# INTRODUCTION TO COMPUTER CONTROL



# 1



Before we begin the detailed discussions of computer control that start in Chapter 2, let us take some time to explore the meaning of the term "computer control." To some, this term may call up images of futuristic robots, huge, automated factories, and complex spacecraft. To others, computer control may seem like something that only scientists use—inevitable, but too complicated for them to understand. In fact, scientists and industrial designers *do* use computer control in spacecraft and automated factories, and these applications are quite complex, but the term can also be applied to much simpler home applications, such as those described in this book.

## 1.1 WHAT IS COMPUTER CONTROL?

The overall objective of this book is to enable you to understand computer control. With this understanding will come new insight, allowing persons not directly involved with scientific applications of the computer to see many ways the computer can be applied in the home. As these home applications of the computer are developed, you will lose whatever fear you may have of automation and gain

respect for what computer control can do and how it can help you. Another major objective of this book is to show that computer control does not have to be complicated.

We use the VIC-20 Personal Computer in this text as the means of control. However, the concept of computer control is applicable to almost any home computer. Further, if you have a VIC-20 at home, you have already used computer control and may not have been aware of it.

To answer the question of what is meant by computer control, we will show several examples of how a home computer is used. The concept of computer control is quite simple, and is graphically illustrated in Figure 1.1. In this diagram the computer is connected in some way to direct the action of another piece of hardware. This, in essence, is what computer control is all about: The computer is directing the physical or electrical action of an external hardware device.

In almost any computer control application, the computer must have a way of understanding how the external hardware is responding to its control. Therefore, the computer must not only direct the

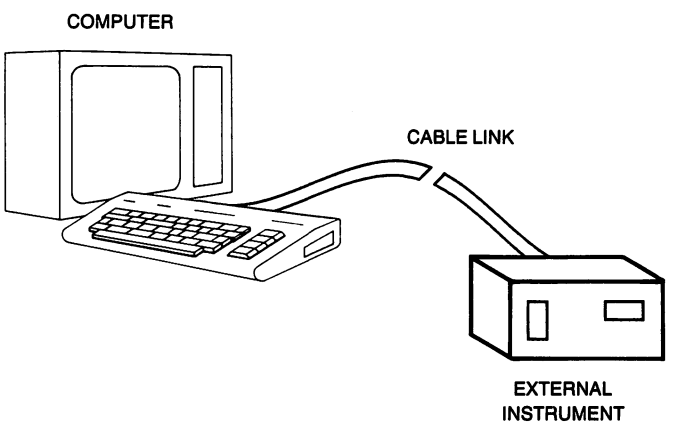



***Figure 1.1**–The main concept of computer control is that the computer will direct the action of an external piece of hardware. The link shown here is made by a cable. However, the link may also be made by transmission at radio frequencies, without a physical connection.*

action of the external hardware, but *monitor* it also. In Figure 1.2 we see that the computer will receive information from the external hardware. Based on that information, the computer can modify the directions it gives to the external device.

This simple example illustrates the basic elements of computer control. The two processes—sending directions to the external hardware from the computer, and receiving information from the external device—are the essential concepts of computer control. At this point in our discussion, it may be valuable to list these two important concepts:

1. The computer sends directions to the external hardware.
2. The computer receives information from the external hardware.

These two ideas are the basis for computer control. The purpose of this book is to explain *how* these two tasks are performed.

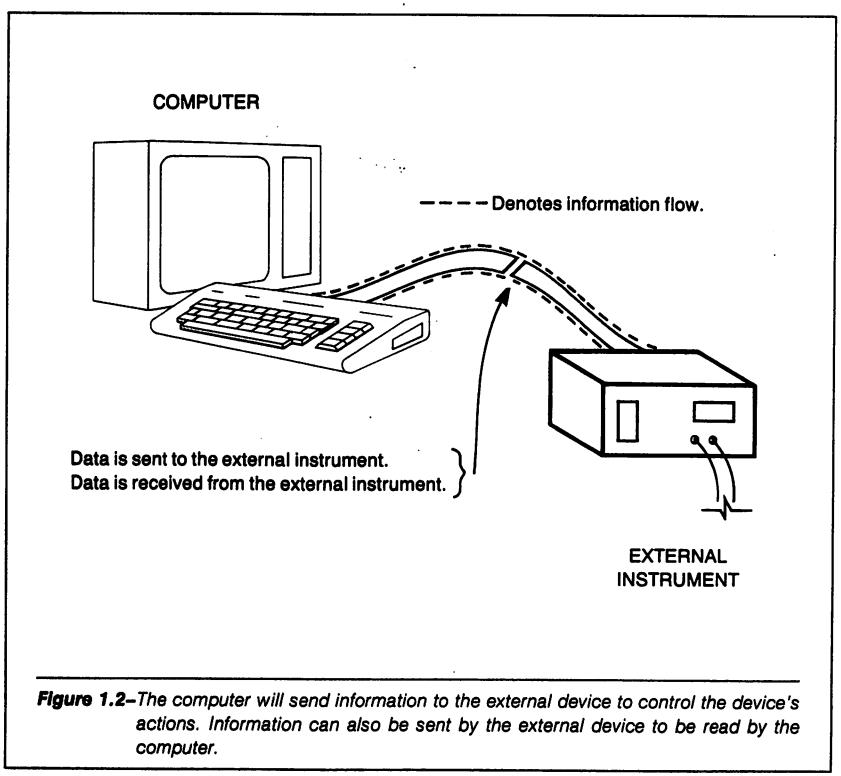



***Figure 1.2**–The computer will send information to the external device to control the device's actions. Information can also be sent by the external device to be read by the computer.*

## 1.2 A PRACTICAL EXAMPLE OF THE TWO BASIC CONCEPTS

An example of computer control that most of us can imagine ourselves using is a home security system. With the help of diagrams, we will see that this goal can be achieved using only the two concepts we have outlined. We will return to this example in Chapter 5 and develop such a system in detail.

Let us start by defining what we want the security system to do. In short, we want our home safe from intruders. Unfortunately, this type of statement is useless for our purpose, because the word "safe" conjures up entirely different meanings to different people. For this example, let us define the function of our security system a little more precisely: the system will detect any window or door being opened. When this occurs, the system will indicate which door or window it is. Finally, the system will sound an alarm if any door or window is open. This is the definition of exactly what we wish the system to do. In Chapter 5 we will expand on this system to show many new ideas for a home security system that is computer-controlled.

Using this definition will require that the doors and windows be capable of sending information to the computer, as shown in Figure 1.3. This will involve the second concept that was given in Section 1.1. That is, the computer receives information from an external device.

If the computer detects that one of the windows or doors has been opened, an alarm must be set off. This is shown in Figure 1.4. In order for the computer to set off the alarm, information from the computer must be sent to the alarm to direct the external hardware to emit the noise. This type of computer action was shown as concept number 1 in Section 1.1, where the computer sends information to an external device.

Looking at Figures 1.3 and 1.4, we can see that all of the functions we require our system to perform can be done by the computer using only the two concepts outlined. It is true that we have, so to speak, "waved our hands" over some important points. For example, *how* does a door or window send electrical information to a computer? *How* does a computer sound an alarm? These points were deliberately ignored. It is possible for a door or window to be made to send electrical information to the computer and for the computer to sound an alarm. *How* that is done is what this book is all about. We will

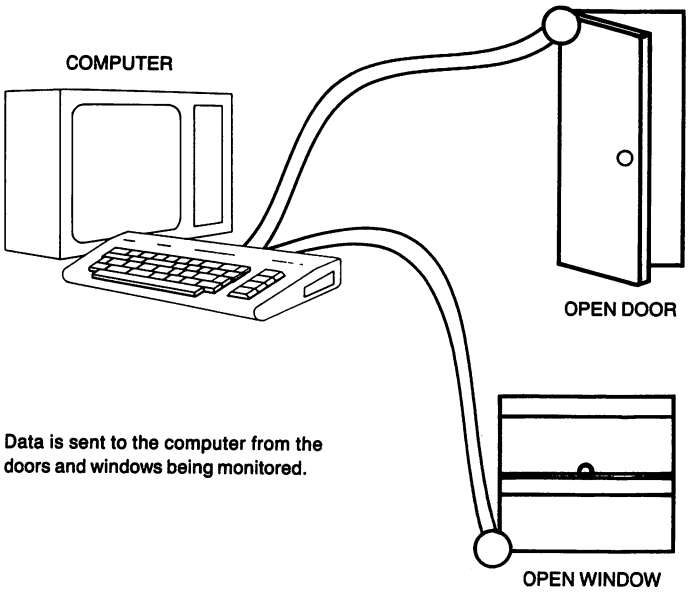



***Figure 1.3**–Doors and windows send information to the computer to report their status, either "open" or "closed."*

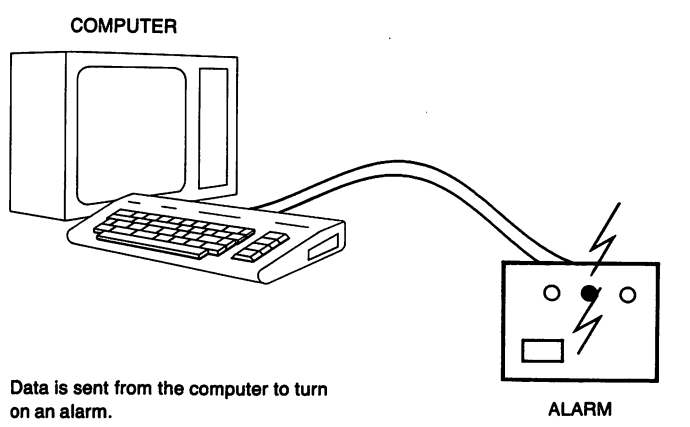



***Figure 1.4**–A computer will send information to the alarm to inform the user that a window or door has been opened.*

show exactly how to do these things in later chapters. It is too early in the discussion to approach these details yet; we must first understand where we are heading. From our first example, we can see that the entire subject of computer control can be reduced to the repeated application of the two basic concepts. These two concepts are what should be understood at this time. You may be assured that this text will cover in *detail* how to achieve the points glossed over in this first example.

## 1.3 SOME NEW VOCABULARY

When we enter a new area of study, a major obstacle we face is learning the vocabulary essential to the area. Interfacing a computer so as to control an external device is no exception. This section will discuss some of the new words that you are likely to encounter when reading or talking about the topic. If you are reading this book, and own or have access to a VIC-20 computer, you are probably also beginning to read some of the magazines and books now available on the subject. You may also have friends or associates who talk "computerese." All the new terms you may encounter cannot be defined here, but as the topics in the text require it, new vocabulary will be introduced. The new words defined in this section are meant to help the beginner to understand the literature (including the VIC-20 documentation) as quickly as possible.

The two major concepts we have introduced, sending information from a computer to an external device, and receiving information from an external device to be used by a computer, are referred to as *output* and *input* respectively. These terms are applied both to the act or event of communicating information between a computer and an external device, and to the information itself. Thus, output is both the transfer of information from the computer to the external device and the information sent, and input is both the transfer of information from an external source into the computer, and the information entered.

If you have ever used a VIC-20 Personal Computer, you have made use of input and output. Whenever you press a key on the keyboard, the computer is inputting the information you send. When the letter

you pressed on the keyboard appears on the screen of your video monitor, the computer is outputting information. We do not usually think of the VIC-20's keyboard or video monitor as devices for input and output, because they are integral parts of the computer system. In fact, although these components are part of the system, they are external to the computer's *central processing unit,* or *CPU.* Ordinarily, we think of input and output as being performed through external devices connected to the computer by, say, a cable. These devices might include the special applications we will be developing in this book, or commercially available *peripherals.*

Consider the case of a printer attached to a system. When the VIC-20 computer gives a printout, the printer is operating under the control of the computer. During the time the printer is printing and causing the paper to scroll, the computer is outputting information to the printer. If you have a disk-drive or cassette tape recorder attached to your VIC-20, the computer is controlling these devices. When the computer writes your program onto the disk or the tape cassette, it is outputting information. When the computer reads a program from the floppy disk or the tape cassette, it is inputting information.

Figure 1.5 illustrates the general concept of input and output. These two terms are sometimes combined and abbreviated to the single term *I/O.* Whenever the computer is controlling an external device or performing input and output events, it is said to be "doing I/O."

The next term we will discuss is *transducer.* This word is often used in discussing computer I/O. To illustrate what a transducer is, let us return to our example of a home security system. In this case we constructed a means by which the computer would receive input information concerning the physical position of a door or a window. The information that would be input to the computer is electrical. The door or window gives out information only in the form of physical movement. That is, all the door or window can do is move. We therefore need some type of device that changes the physical movement of the door or window into electrical information that can be input to the computer. The device that does this is called a *transducer.*

This example is one type of transducer. As we will see in Chapter 5, the device used is a simple switch, open when the window is open and closed when it is closed. In general, a transducer is a device or piece of hardware that produces an electrical output when given a physical input, translating one form of energy or information into

another. In the example just given, the transducer must be capable of converting movement into electrical information. Other examples of transducers are a temperature transducer, a pressure transducer, or a revolutions-per-minute transducer. As you can see, transducers are classified by the type of information they translate into electricity. Figure 1.6 shows how a transducer would fit into the input path between a computer and an external source.

At this time you may wonder if a transducer is used when the computer is outputting information. In the example of the home security system, the computer would sound an alarm if an open window or door were detected. The alarm would be a loud bell or siren. This type of device receives electrical signals from the computer and produces a physical effect—the motion of air, or sound. Broadly, any

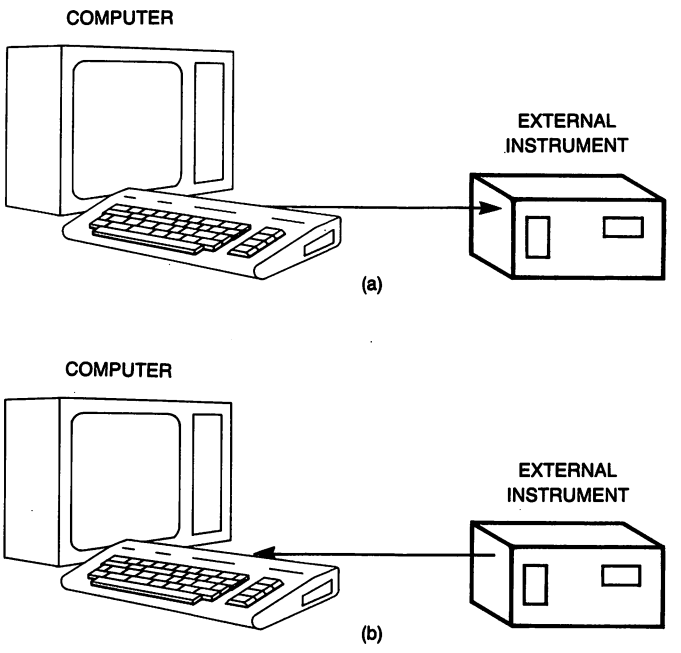



***Figure 1.5**–a) This is an example of* output. *The computer is sending information to the external device.*
*b) This is an example of* input. *The computer is receiving information from the external device.*

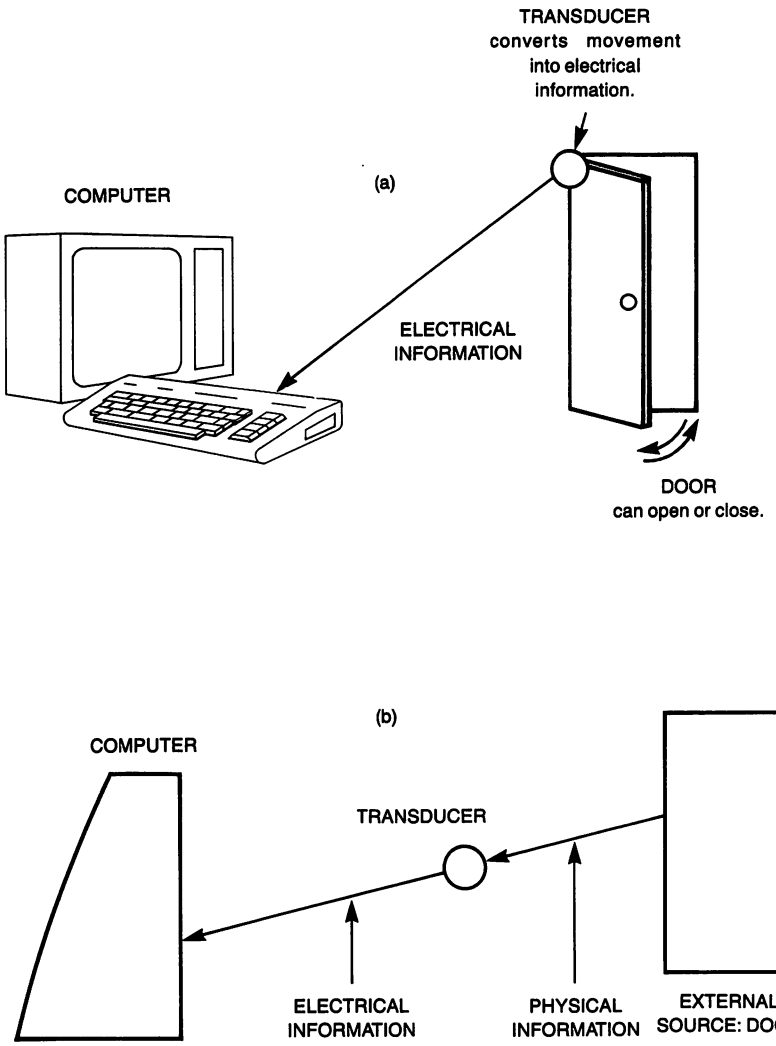



***Figure 1.6**–a) A transducer will transform the physical movement of the door into electrical information, which can then be input to the computer.*
*b) This block diagram shows where a transducer fits in the computer input path. The transducer will have a physical event input to it. That is, it will measure or record a physical event. The output of the transducer will be an electrical signal related to the physical event being input to it.*

device that translates one form of energy into another (including electricity into sound) can be called a transducer. In the context of computer control, however, these types of devices are not generally classified as transducers. Instead, they are grouped together generally as *output devices,* or simply known by their specific names: bell, motor, relay, fan, heater, and lights. Figure 1.7 shows how a relay is used to activate these devices to output information from the computer.

The next term to discuss is *digital.* Many people are already familiar with this particular word. In a general sense, the term means that there are discrete values that a particular event can equal. For example, the television channels are assigned certain frequencies, out of an infinite number of possible frequencies. We can say that selecting a TV station is a digital process, because the channel values can only be those specified and nothing in between. We do not have channel 2.5, for example, we have channel 2 or 3.

The term digital is usually contrasted with the term *analog.* Digital

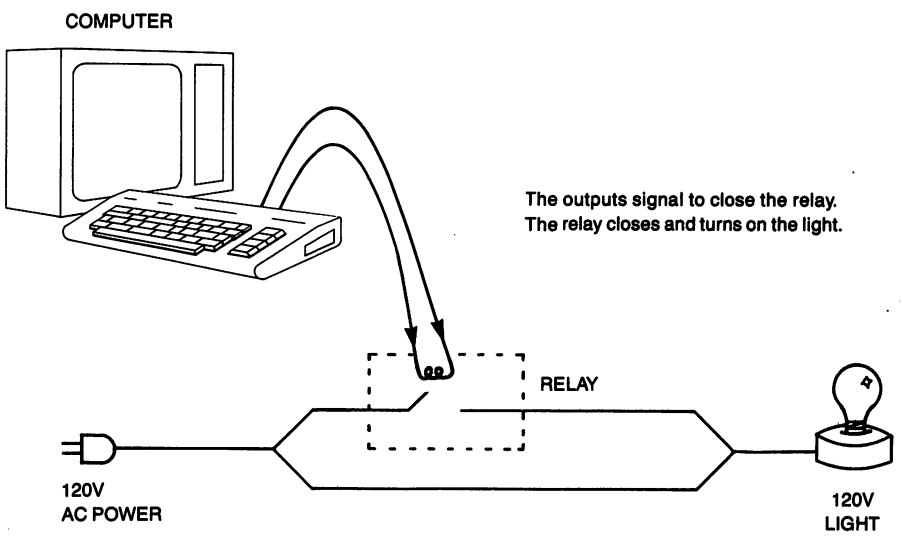



***Figure 1.7**–A physical device that will transform the electrical output of the computer into a physical event. This device, a relay, will transform the electrical output of the computer into movement of the relay contacts.*

systems count discrete units; analog systems measure over a continuous range. This topic is discussed at greater length in Chapter 7.

When applied to a computer, the term *digital* means there are discrete voltage levels present in the information. All information sent out (or output) by the computer or received (input) by the computer must be made up of two voltage levels. All information that is used by the VIC-20 consists of these two voltages. This is the reason the VIC-20 is called a *digital* computer. There are discrete values for the information; further, there are only two discrete values.

The two voltage levels for the information used in the VIC-20 are given the labels 0 and 1. These are the only choices each digit may take in base 2, the binary notation used in almost all computers. All information processed by a computer is expressed in combinations of these two digits. Also, a logic in which every statement is either "true" or "false" is a *binary* logic. In computer logic, the digits 0 and 1 are assigned to these two values, and are called "logical 0" and "logical 1." For these reasons, the two terms "binary" and "digital" are often used almost interchangeably in the computer field. In digital electronics, the actual voltage value of a logical 0 or 1 may vary among different types of digital equipment. In some digital systems, a 0 may be equal to −1.9 volts and a 1 may be equal to −1.5 volts. In other digital systems, a 0 may be equal to 0.0 volts and a 1 equal to 9.0 volts. The voltage values for the VIC-20 are called TTL (Transistor-Transistor Logic) voltage levels.These voltage levels are approximately 0.0 to 0.8 volts for a logical 0, and 2.4 to 5.0 volts for a logical 1. We will discuss exactly what is meant by these voltage levels in Chapter 4. For now it is important to understand that there are only *two* distinct voltage levels present in the VIC-20 when information is output or input.

The next term to discuss is *data.* This term is used in many different ways when discussing digital computers. For our purposes the term *data* will refer to the digital information that will be input to, or output from, the VIC-20 computer. Physically, data in a computer can be thought of as a series of electrical pulses occurring at specified times.

Since the information processed by the VIC-20 is digital, it can be referred to as *digital data.* That is, the VIC-20 will output and input digital data only. No other type of data is acceptable. If you are a beginner in computers, other types of data may not occur to you at this point. However, as we proceed in this text, we will see other meanings for the term *data.*

Let us now discuss the term *bit*. To define this term, we must first look at how digital data will appear in the VIC-20. Figure 1.8 shows what one typical piece of data would look like to the VIC-20. The data is composed of eight single 1s, 0s or any combination of the two. Each unit of digital data is called a bit. The word is short for "binary digit." There are eight bits in the data shown in Figure 1.8.

The next new term is *byte*. A byte consists of eight bits that are communicated at the same time; the bits are said to be in *parallel*. We can therefore describe the digital data output and input by the VIC-20 as data bytes. The VIC-20 computer will output and input digital data in units of eight bits. This information will become important when we discuss how to select exactly how many 1s and 0s will be present in each data byte. Figure 1.9 shows some different data bytes.

As we will see, the position of bits in a byte is significant, because a byte represents a number in base 2. Just as the position of numerals in a base 10 number determines their values as powers of 10, the position of numerals in a base 2 number (or bits in a byte) determines their values as powers of 2. For example, in the base 10 number 357, the

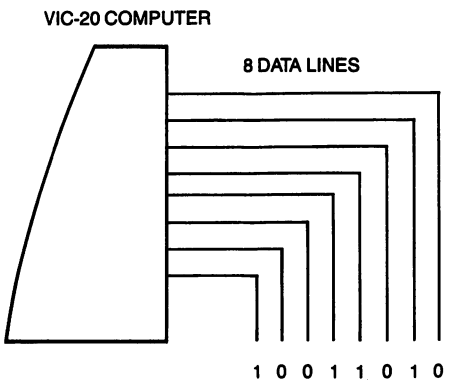



***Figure 1.8**–Data output lines can be set to any combination of logical 1s or 0s.*

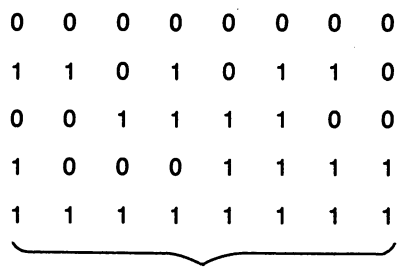



***Figure 1.9**–Examples of different data bytes. Each byte consists of a combination of eight digits, either logical 1 or logical 0, called bits.*

numeral 3 represents $3 \times 10^2$. In the number 35, the same numeral represents $3 \times 10^1$. Likewise, in the base 2 number 100, the numeral 1 represents $1 \times 2^2$, and in the base 2 number 10, it represents $1 \times 2^1$. In both number systems, the rightmost numeral represents a power of zero. As data bytes, all base 2 numbers are filled out to eight places by adding zeroes. Our examples above would thus be represented as 00000100 and 00000010. Of these eight bits, the leftmost bit represents the highest power of 2 (or *weight*), $2^7$, and is called the *most significant bit (MSB)*. The rightmost bit, $2^0$, is called the *least significant bit (LSB)*.

Sometimes, when discussing data input and output, people will refer to the data as "data words." A data word is the number of bits the computer treats as a unit and processes simultaneously. For the VIC-20 and other *eight-bit* computers, the data word will be eight bits, and so data *byte* and data *word* are synonymous. For other types of computers, the data word may be larger than eight bits.

Another term that is used often in computer interfacing and computer control is *nibble*. A nibble consists of four bits of digital data. We won't be discussing data as nibbles in this book, but you may run across the term in the literature. Figure 1.10 shows some examples of data nibbles. In the VIC-20 , the data byte will consist of two parallel nibbles, as shown in Figure 1.11.

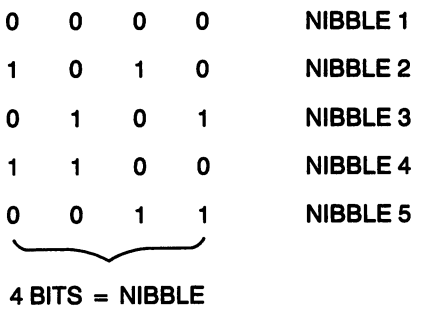



***Figure 1.10**–Examples of different data nibbles. Each nibble consists of four bits.*

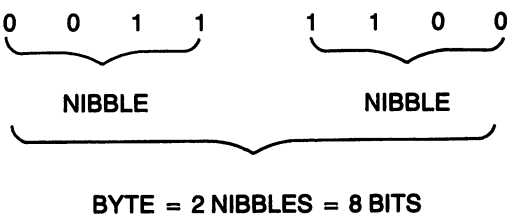



***Figure 1.11**–A data byte consists of two parallel nibbles.*

## 1.4 SUMMARY

In this chapter we have introduced the topics that will be covered in this text. The chapters to come will show how to correctly input and output information from the VIC-20 in order to control external devices. We next discussed the two basic concepts of computer control, using the example of a home security system. In this example it

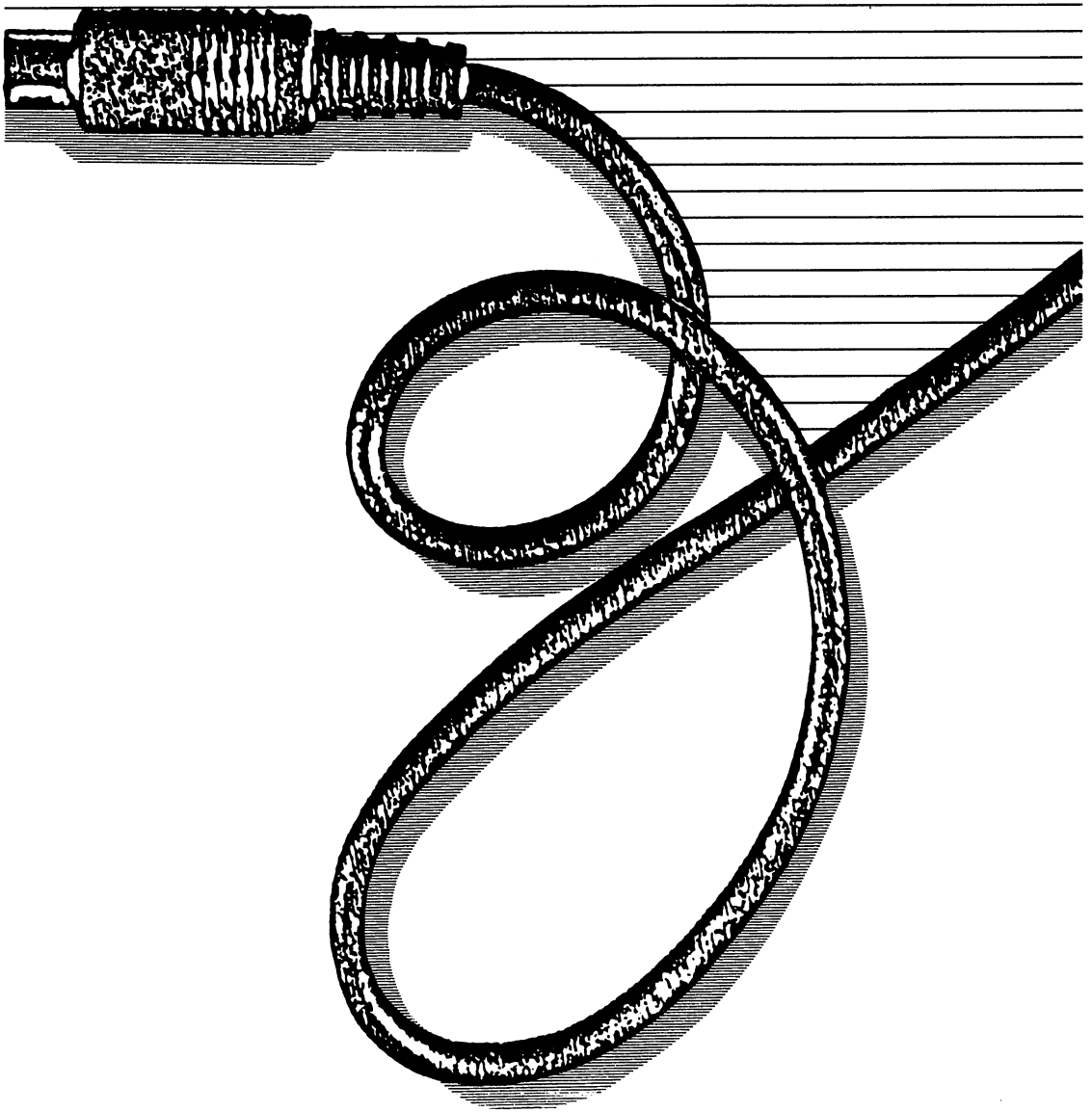
was shown that the computer must send information out to the external device, and it must be capable of getting information back from the external device.

The next section of the chapter was devoted to introducing some new vocabulary that is used when discussing computer control. This vocabulary is used in the industry and in the literature. It is presented here to help you learn the language that is common to computer control. We will use most of these terms in this text. The definitions given are summarized below for your convenience.

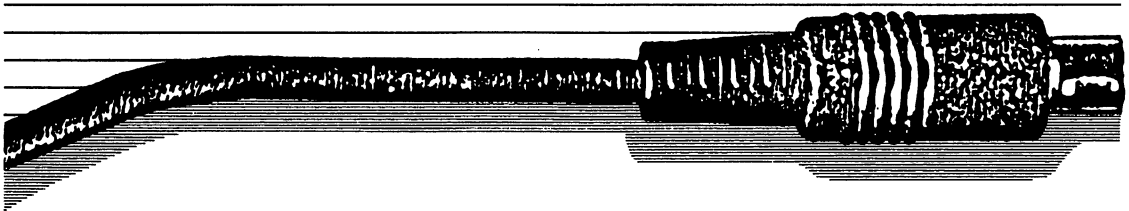
## Words Defined in This Chapter

1. *Input:* The computer receives information from an external device.
2. *Output:* The computer sends information to an external device.
3. *Transducer:* Any device that converts a physical event into electrical information.
4. *Digital:* Having discrete output values. The selection of a television station is digital, because the channels that can be tuned in have a certain preset value. In a digital computer there are discrete voltage values for the information.
5. *Binary:* Having two possible states or values. In a digital computer there are only two voltage levels associated with the information processed. *Binary* refers to this fact.
6. *Data:* This is the information that is output, input, or processed by the VIC-20.
7. *Bit:* A bit is one of the eight places in a digital data word used by the VIC-20 computer. A bit will be either a logical 1 or a logical 0.
8. *Byte:* A byte consists of eight parallel bits.
9. *Nibble:* A nibble consists of four parallel bits. Two parallel nibbles equal one byte.

# SOFTWARE FOR OUTPUT FROM THE VIC-20 COMPUTER



# 2



In this chapter we will discuss the programming necessary to output digital information from the VIC-20 computer to the outside world. We start the discussion assuming the reader is familiar with VIC BASIC, the version of the BASIC programming language used on the VIC-20. No other assumptions are made. Each new topic will be clearly discussed, and examples will be given.

The examples given in this chapter are designed to be used with the Creative Microprocessor Systems (CMS) I/O system. The CMS I/O board is a visual means of testing your VIC-20 input and output programs. It will install directly into the VIC-20, and data can be output and input with it. You do not have to purchase this board to benefit from this chapter, but you will learn more about writing this kind of program if you use the CMS I/O board (or an equivalent device) while studying the examples given. Information regarding availability of the CMS I/O system for the VIC-20 is given in Appendix D.

## 2.1 INSTALLING THE CMS I/O SYSTEM

Before we start learning the software required for inputting and outputting data with the VIC-20, we must first learn how to install the

hardware into the computer. This discussion will bring out some important general information about the physical connections necessary to perform computer control with the VIC-20.

The CMS I/O system comprises two printed-circuit boards and an interconnecting cable. One of the circuit boards plugs directly into the VIC-20; the second circuit board connects to the first board via a 24-pin ribbon cable. See Figure 2.1 for a diagram of these two boards. Let us now go over the details of how to connect the first board directly to the VIC-20 computer.

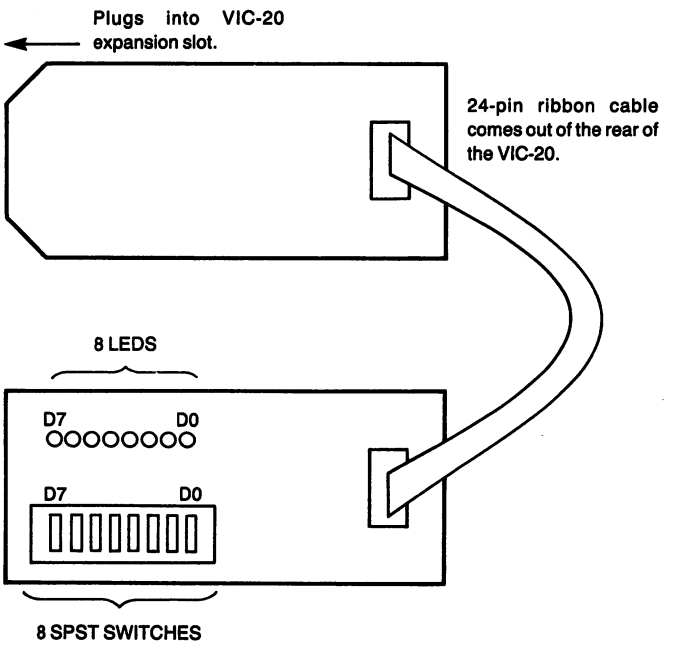



***Figure 2.1**–Diagram showing the two printed-circuit boards that constitute the CMS I/O system. One board plugs directly into the VIC-20, while the other board sits alongside the computer for easy viewing by the user.*

Figure 2.2 shows the expansion connector for a VIC-20 Personal Computer. It is through this connector that external instruments are controlled by the VIC-20. The VIC-20 also has a connector labeled "USER PORT." We will *not* use this connector, because using the expansion port will show a more general technique for connecting your computer to the outside world.

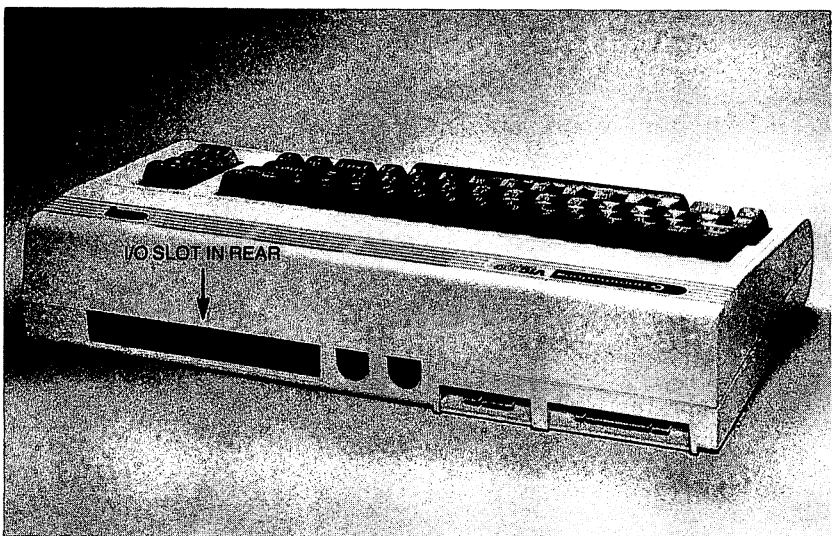



***Figure 2.2**–Photograph of the VIC-20 computer with the I/O connector shown in the rear.*

It should be mentioned before we begin that static electricity is easily generated, and can seriously damage a computer's circuitry. Rodnay Zaks' *DON'T! (or How to Care for Your Computer)* (Sybex, 1981) describes ways of avoiding this problem.

The procedure for installing the CMS I/O system is as follows:

1. Turn off the power on your computer and any other peripherals. Always turn off the power when installing any hardware into the computer.
2. Place the VIC-20 with the rear of the computer facing you to expose the expansion I/O connector. This is a 44-pin, 22/22 (22 pins on top, 22 pins on bottom) connector. The connector is located on the left with the rear of the system facing you.
3. Install the 24-pin ribbon cable into the 24-pin socket on the

CMS I/O board labeled "CMS VIC-201." It is important to install the 24-pin connector with pin 1 in the correct place. See Figure 2.3 for a diagram showing how to physically install the cable into the respective circuit boards.

4. Install the circuit board labeled "CMS VIC-202" into the expansion connector. It is important to install the circuit board correctly, so that the ICs (integrated circuits) are on top. Figure 2.4 shows a diagram of how the circuit board is to be physically installed in the VIC-20.
5. Run the ribbon cable around the back of the VIC-20. The cable will then connect to the second circuit board, labeled "CMS VIC-202."
6. Connect the remaining end of the 24-pin ribbon cable into the second circuit board. Care should be taken to insure that pin 1 of the cable is connected to pin 1 of the socket on the second circuit board.
7. At this time your system will appear as shown in Figure 2.5. Your computer is now ready for use with the CMS I/O system.
8. Turn on the system power.

## 2.2 THE POKE INSTRUCTION

It was stated earlier that the reader is assumed to be familiar with BASIC. You are not expected to be an expert programmer, but you should know enough to be able to write simple BASIC software and run programs using the VIC-20 computer. We will introduce new information based on that assumption.

To get started, you must know how to direct digital information from a BASIC program out to the CMS I/O board. If you can direct information to the CMS I/O board, then you can direct information to any type of I/O board. The CMS I/O system is used as a tool to promote understanding. After this initial discussion, several examples and problems will be given. They are designed to let the reader get "hands-on" experience using the software necessary for outputting information from the VIC-20. The CMS I/O board will allow the user

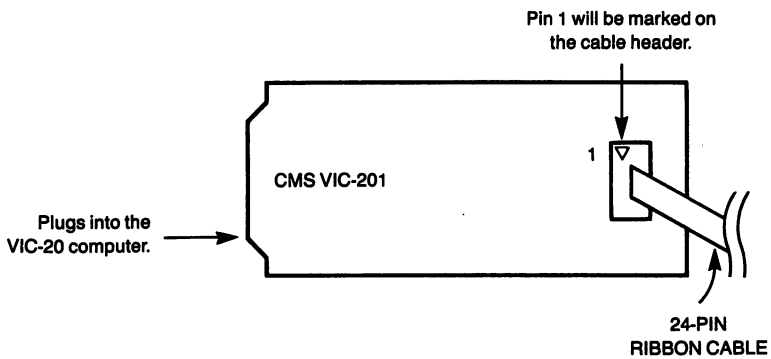



***Figure 2.3**–The CMS VIC-201 board will plug directly into the expansion connector in the rear of the computer. A 24-pin cable will connect to this printed-circuit board.*

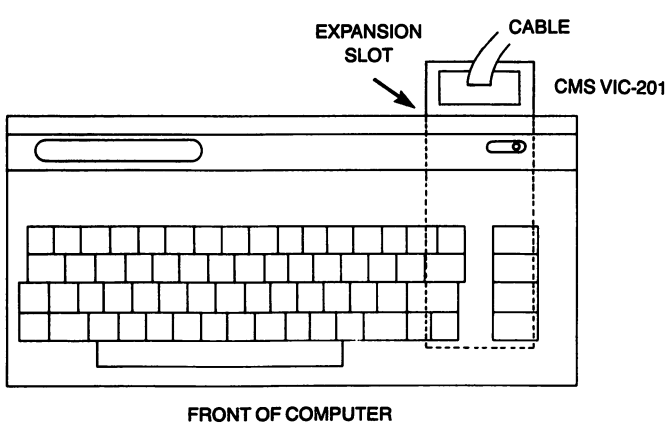



***Figure 2.4**–Pictorial diagram showing exactly how the CMS-201 board is installed into the VIC-20.*

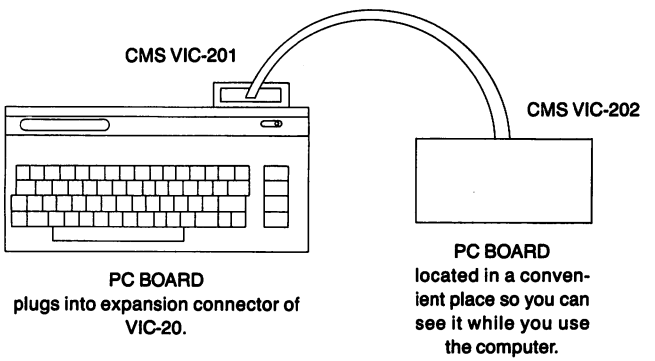



***Figure 2.5**–When both boards are connected properly, the complete system will appear as shown.*

to instantly verify whether or not the software written is operating correctly.

## Let's Get Started

To output digital information from a BASIC program, we will use an instruction that may be new to some VIC-20 computer users: POKE. A full description of the POKE instruction is given in the VIC-20 user's manual, which comes with your system when you purchase it. We will describe this instruction in enough detail here so that you will feel comfortable in its use.

The form of the POKE instruction is:

**POKE address, data**

Let us discuss the two parts of the POKE instruction, *address* and *data*.

The address used by the POKE instruction will indicate the physical space in the computer where the information will be sent out from the program. More than one number may be specified for a POKE address. The VIC-20 has over 65,535 memory addresses. However, the complexities of I/O addressing are beyond the scope of this book.

Therefore, we will use only a few of the 65,535 available address numbers in this text. Once you understand how to use these addresses, it will be much easier to understand how other POKE addresses are used.

An address in computer programming can be likened to the address of a house. The only way the mail carrier knows where to deliver a letter is by matching the address on the envelope with the address on the house. To extend this analogy, think of the address on the envelope as the address specified in the POKE instruction. The address of the house will be the address we specify for our CMS I/O circuit. A VIC-20 PC will match the address of the "envelope" (POKE address) with the address of the "house" (I/O slot address) and deliver the information. (See Figure 2.6.)

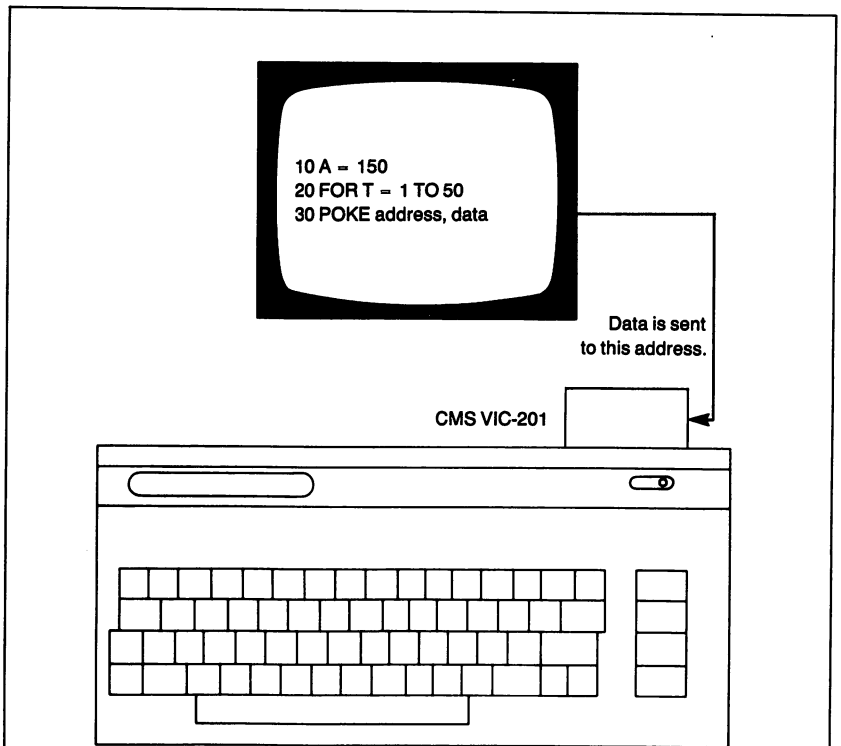



***Figure 2.6**–When a POKE instruction is used, the address specified will select the expansion connector for communication.*

The second element in the POKE instruction is the data. This will be the actual digital information to be sent to the address specified. In our analogy of the address and the mail carrier, we may think of the data as the actual letter that was delivered.

The address specifies where in the entire system to send the information. The data is the information. With this broad overview of the POKE instruction, let us get into some specifics.

## 2.3 FORMING THE ADDRESS FOR THE POKE

We have discussed the concept of the address in the POKE instruction, but we have not shown exactly how to use the address. In this section we will show how to determine the correct address. It should be noted that determining an address can be much more complicated than we will show. However, if you are just starting to learn the interfacing of a computer, you will find this introduction to be a good entry point. What we will show here will work and work well. Only when you begin to tackle more sophisticated interfacing problems will a greater understanding of address calculation be required.

In the expansion connector on the VIC-20 there are two I/O enable lines, labeled $\overline{I/O2}$ and $\overline{I/O3}$. Each of these two lines has 1024 unique addresses associated with it. $\overline{I/O2}$ has addresses from 38912 to 39935, inclusive. $\overline{I/O3}$ has addresses from 39936 to 40959, inclusive.

Using this information, we can select the proper address for the POKE instruction simply by knowing what the address is on the CMS I/O circuit. This is true of any POKE instruction. The hardware we wish to communicate with must have an output address or addresses. In order to send information to this hardware we must know the proper address. The CMS I/O circuit will connect to the $\overline{I/O2}$ line of the expansion connector. We will use the first available address that asserts this enable line, 38912, as the POKE address of the CMS I/O system.

To send data to the CMS I/O circuit you would use the instruction:

```
POKE 38912,data
```

Notice that in this example of the POKE instruction, only the address was specified. The content of the data is not important to us yet. The data will be sent to whichever output device "resides" at the

specified address. In this case, it is the CMS I/O board.

You may want the flexibility of using your program regardless of the output address of the hardware being communicated with. If your BASIC program includes a POKE statement with a specific address, then that address will need to be altered if the address of the hardware changes. Remember that the hardware address and the POKE address must match for proper communication.

To allow your program to run when the address of the hardware circuit changes, you can let your program determine the POKE address. This could be done by following the flowchart shown in Figure 2.7. The BASIC program prints a message asking the user which I/O address the user wishes to communicate with. Based on that answer, the POKE instruction sends data to the appropriate address.

Figure 2.8 shows one way a program could be written to realize the flowchart of Figure 2.7. Notice that the address of the POKE instruction has been made a BASIC variable, S1. After the address has been

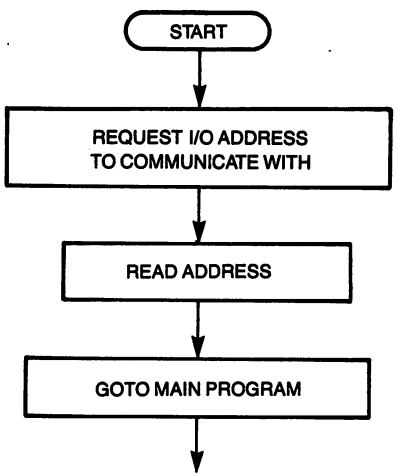



***Figure 2.7**–Flowchart showing the major steps involved in entering the POKE address as a BASIC variable.*

entered into the program, the form of the POKE instruction would be:

**POKE** S1,data

where S1 corresponds to the address of a particular I/O circuit.

```
10 PRINT "INPUT I/O ADDRESS ";
20 INPUT S1
30 GOTO "MAIN PROGRAM"
```

***Figure 2.8**–BASIC program to enter the I/O address as a numeric variable.*

## 2.4 CALCULATING DATA FOR THE POKE

Before we examine the calculation of the data portion of the POKE instruction in detail, let us discuss exactly what the data will do. The output section of the VIC-20 will use byte (or 8-bit) output. This means that there are eight physical, electrical lines over which information is passed from the VIC-20 to the external output circuits; each of these lines can communicate one bit. (See Figure 2.9.)

It is not really important to know *why* there are only eight lines. To perform computer interfacing and computer control, we simply make use of this fact. When the POKE instruction is executed in a BASIC program, the data that is output is contained in eight bits.

All of the information that can be transferred to the output device in a single POKE instruction is included in the eight bits. This means that you must keep in mind what each of the eight bits being transferred does, whenever you use the POKE instruction to send information from the VIC-20 to an external device.

For example, one of the eight bits might turn on a light, another might sound an alarm, yet another might open a door, etc. Further, all eight bits may be used together to form a unique combination to which the output device may respond. To use the POKE instruction, you must be able to generate any information you want, by controlling each of the eight bits.

We will concentrate, in these early discussions, on showing how to set any bit that will be used in the POKE instruction to a logical 1 or a logical 0. These terms were introduced in Chapter 1, but if you are

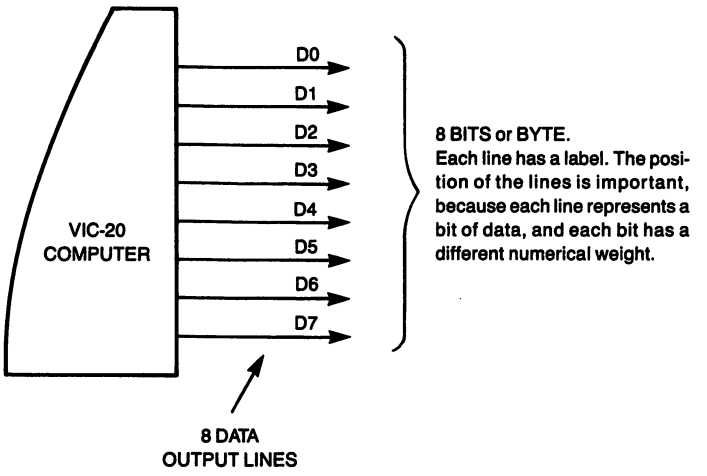



***Figure 2.9**–Block diagram showing that there are eight physical output lines over which data will be transferred from the VIC-20 computer to the external output device. The output lines originate from the expansion connector of the VIC-20.*

new to the field of digital logic, they still may not mean much. However, if you are interested in learning how to electrically control hardware with a computer, then you must become aware of exactly what these terms mean. For now, we will define logical 1 and logical 0 as precisely as possible, and use the terms in our explanation. When the examples of outputting information are given, you will have a good idea of exactly how to set any of the eight bits to a logical 1 or a logical 0. This will be true even if you do not yet fully understand exactly what a logical 1 or logical 0 means to the external device.

Logical 1 and 0 can be defined by the ranges of voltage levels they correspond to. These definitions will apply to the VIC-20, and to most other home computers. A logical 1 is a voltage level greater than 2.4 volts and less than 5.0 volts. Any time a digital line is set to a logical 1, the voltage on that line will be within these limits. A logical 0 is a voltage level of less than 0.8 volts and greater than 0.0 volts.

These definitions will help you relate physical actions to the logical definitions we will deal with in computer control. For example, suppose we say that all eight bits or lines are a logical 1. This means that all of the lines are set to a voltage that falls within the range specified for a logical 1.

With this brief introduction, let us discuss how any of the eight output lines can be set to a logical 1 or a logical 0 under software control. There are two key facts to keep in mind at this time. First, we must remember that all eight bits are output to the VIC-20 expansion connector at the same time. That is, all eight bits are output in "parallel." This was illustrated in Figure 2.9. The other important point to note is that the position of the lines within the 8-bit parallel output is significant. The lines are labeled D0–D7. The D stands for data, and the number 0–7 stands for the position of the line within the eight parallel bits. D7 is the most significant bit (MSB), and D0 is the least significant bit (LSB).

Using these two pieces of information, let us examine the eight bits or lines. Remember from Chapter 1 that when eight bits are taken together at the same time, they are called a "byte." We will use this definition throughout the remainder of the text. In a single byte there are eight individual bits.

Since the bits are parallel, or independent of each other, we can use software to set any of the bits to a logical 1 or a logical 0, regardless of the logical state or value of any other bit. In fact, we *must* be able to do this to achieve computer control. To accomplish this, the software must have some means of logically setting any bit to a 1 or a 0. Each bit position in the byte is assigned a number or weight equal to a power of 2. (Remember that a byte represents a binary number.) Each bit position *n* is weighted $2^n$. The weight of D0, 1, is equal to $2^0$. The weight of D7, 128, is equal to $2^7$. The bit positions and their corresponding weights are shown in Figure 2.10.

| Bit Position | Weight |
|---|---|
| D0 | 1 |
| D1 | 2 |
| D2 | 4 |
| D3 | 8 |
| D4 | 16 |
| D5 | 32 |
| D6 | 64 |
| D7 | 128 |

***Figure 2.10**–The position of each bit is assigned a numerical weight, corresponding to its value as a power of 2.*

When we wish to set a particular data bit to a logical 1, the weight of the bit is summed and used as the data in a POKE instruction. For example, suppose bits D0 and D2 were set to a logical 1. The weights of these bits would be summed, giving the following results:

D0=1, D2=4: SUM = 1+4 = 5

The resulting value to be used in the POKE instruction as data would be 5. The POKE instruction would appear as

**POKE address,5**

As we have seen, the address is set according to the specific I/O slot where data is to be sent.

If we were to look at the logical conditions of the data byte to be output with the sum equal to 5, the result would appear as shown in Figure 2.11. Notice, in this diagram, that all the data bits that were used in the summation are set to a logical 1. All other data bits are set to a logical 0.

In short, the smallest data byte is output when no weights are summed; that is, when all the bits in the byte are set to a logical 0. This weight is 0. The largest data byte is output when all the weights are

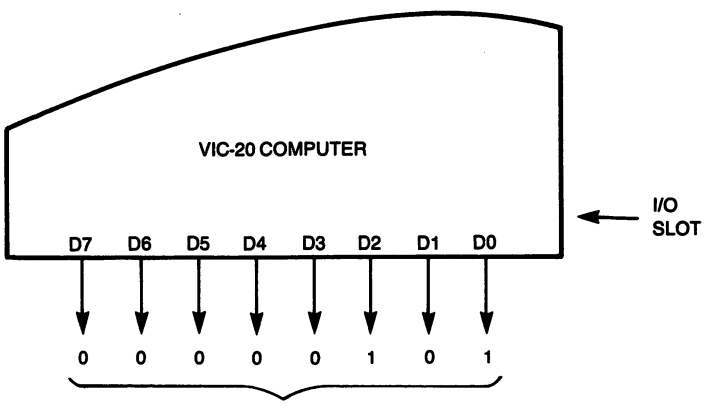



***Figure 2.11**–Diagram showing the logical conditions of the eight output lines that would result if the POKE data were equal to 5.*

summed; when all the bits are set to logical 1. This summation is equal to 1 + 2 + 4 + 8 + 16 + 32 + 64 + 128, or 255. All possible valid weights are within the 0–255 range, and each number represents a unique combination of weights.

Let us try some examples of setting the correct data bits to a logical 1 by summing the appropriate weights. We wish to set data bits D0, D4, and D7 to a logical 1. To do this, the weights of the bits we want are summed: D0=1, D4=16, D7=128. The resulting summation would be 1 + 16 + 128 = 145. A POKE instruction would appear as follows:

```
POKE address,145
```

One more example: Suppose we wish to set the data bits D2, D5, and D6 to a logical 1. We would again sum the weights of the bits we want: D2=4, D5=32, and D6=64. The resulting summation would be 4 + 32 + 64 = 100. A POKE instruction would appear thus:

```
POKE address,100
```

By combining the proper choice of data and the correct address, we can set any bit at any output slot to a logical 1 or logical 0. Further, we can accomplish this using only software. We will discuss the hardware in Chapter 4. For now, you can simply assume the hardware will respond correctly if the software is correct.

Before starting to experiment with the CMS I/O board, let us examine one more aspect of setting the correct data. A program similar to the one that was written to input any address can be written to allow any bit to be set to a logical 1 or a logical 0. A flowchart for this program is shown in Figure 2.12.

Let us examine this flowchart in some detail. The first two steps, (a) and (b), will initialize the variables for the starting sum, V1, and the starting weight, N1. The flowchart then enters a loop, starting with step (c). The loop variable will be I. In step (d) the program will ask the user for the logical value of each data bit. If a bit is to be set to a logical 1, then its weight will be added to the variable V1 in step (g). The result of the addition is stored in the current value of V1. In step (h) the starting weight (N1) is multiplied by 2. This will set the weight equal to the next bit to be tested. After the final value of V1 is computed, the POKE instruction will appear as:

```
POKE address, V1
```

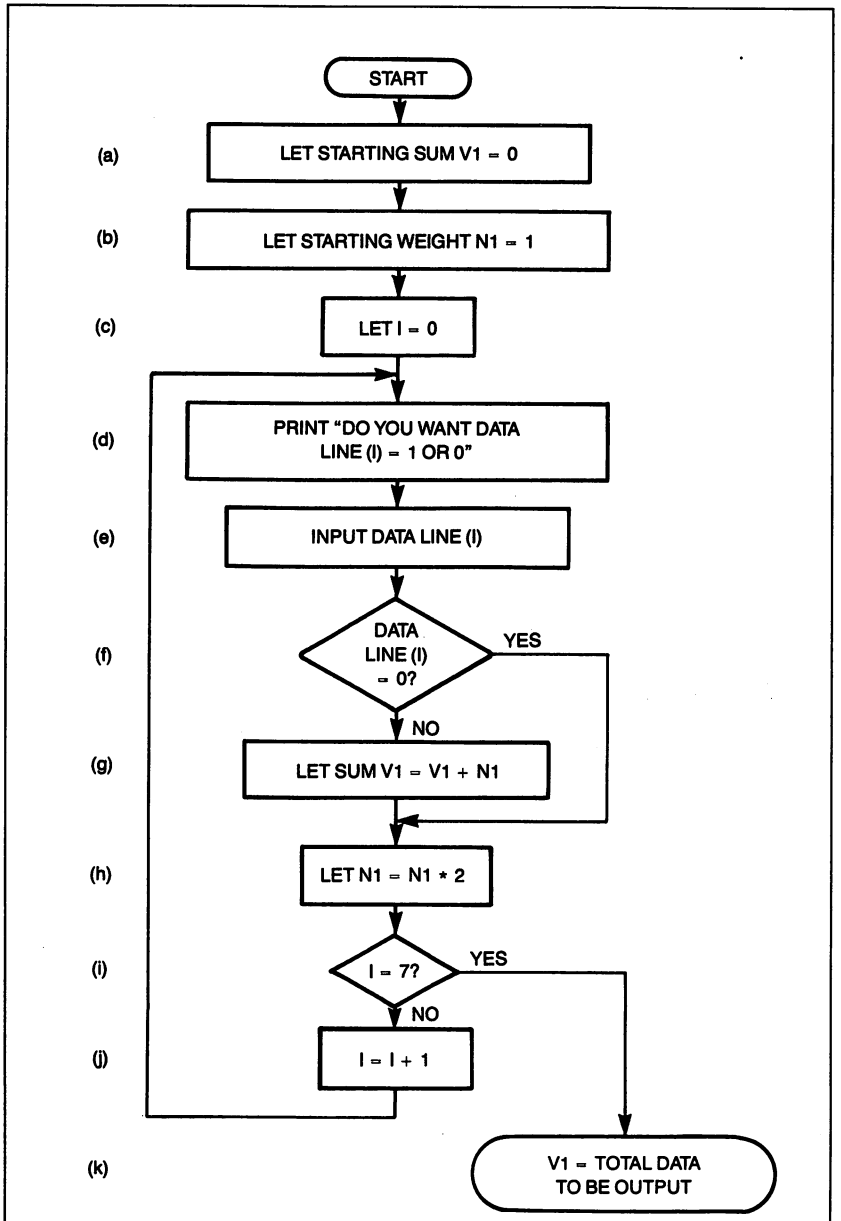



***Figure 2.12**–Flowchart showing the sequence of events necessary to set any of the data output lines to a logical 1 or a logical 0. The sequence begins with D0 (the lowest weight) and ends with D7 (the highest weight).*

A program to realize the flowchart of Figure 2.12 is given in Figure 2.13.

```
10  V1 = 0
20  N1 = 1
30  FOR I = 0 TO 7
40      PRINT "WHAT IS THE VALUE OF D";I;" 1 OR 0"
50      INPUT D(I)
60      IF D(I) = 0 THEN 90
70      V1 = V1 + N1
80      N1 = N1*2
90  NEXT I
```

***Figure 2.13**–BASIC program to realize the flowchart given in Figure 2.12.*

## 2.5 EXPERIMENTS WITH THE CMS I/O SYSTEM

In this section, we will actually write and execute programs that perform output using the POKE instruction. These programs are designed to show exactly how to use the POKE instruction with the VIC-20. All of the following experiments assume that the user has installed the CMS I/O system into the VIC-20 according to the procedure described at the beginning of the chapter.

The CMS output board (VIC-202) has eight light-emitting diodes. These LEDs are labeled D0, D1, D2, D3, D4, D5, D6, and D7. (See Figure 2.14.) When an LED is lit, it indicates that the corresponding bit position in the data byte was set to a logical 1. When an LED is not lit, the corresponding bit position is a logical 0.

For example, Figure 2.15 (a) shows a diagram of three LEDs that are lit. (The darkened LEDs are lit.) In this diagram, bits D3, D5, and D6 are set to a logical 1. The data byte would appear as shown in Figure 2.15 (b).

## 2.6 EXAMPLE 1: LIGHTING A SINGLE LED

In this first "hands-on" example of VIC-20 output, we will write a program that will turn on any of the eight LEDs located on the

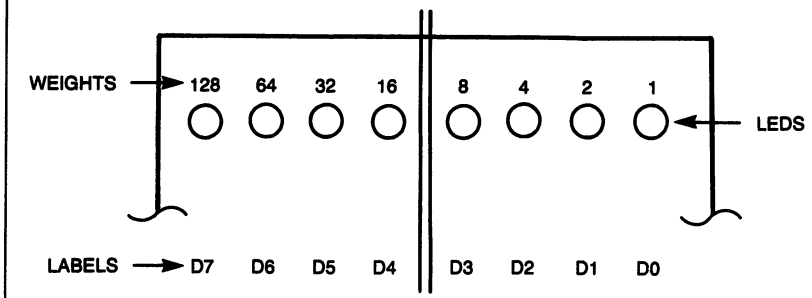



**Figure 2.14**–*The output section of the CMS I/O board has eight LEDs. The weight and data-line label of each LED are also shown.*

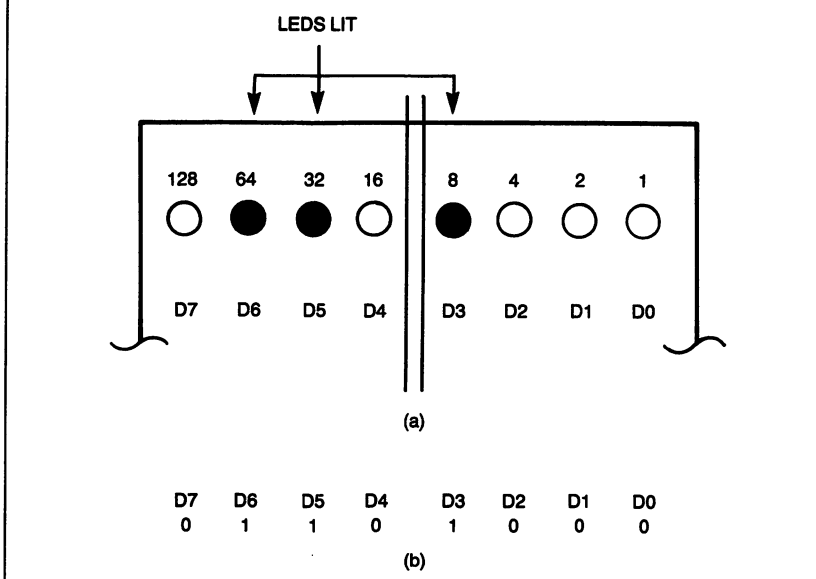



**Figure 2.15**–*a) LEDs D6, D5 and D3 are lit. All other LEDs are off. The number input to produce this combination was 104.*

*b) Corresponding logical levels of the data lines needed to produce the LED pattern shown in (a).*

CMS I/O board. The program we are about to write will assume that the output address is 38912. With this assumption, the program will then ask you which LED you want lit. Finally, the program will light the selected LED on the CMS I/O board.

A flowchart for the program is shown in Figure 2.16. Let us discuss the steps in the flowchart. After this discussion we will show the actual program listing.

Steps 1,2. The user will enter the correct output address.

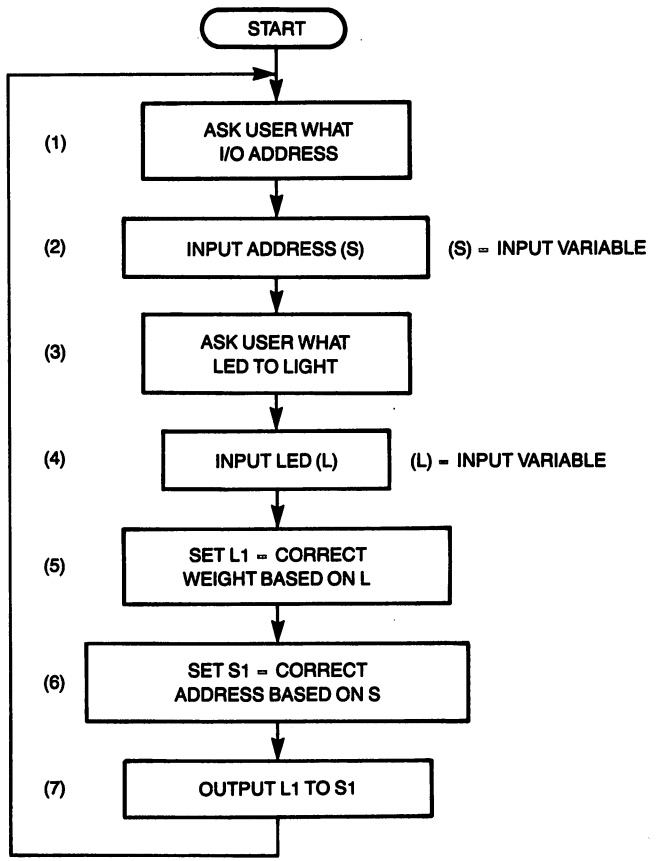



***Figure 2.16**–Flowchart for inputting the I/O address and the LED to be turned on at the external I/O device.*

Step 3. The program will write a message asking which LED to light.
Step 4. The user will input the LED number.
Steps 5, 6, 7. In these steps the program will set the output byte and address to the correct value corresponding to the LED that the user wants to light.
Step 8. The software will now output the correct byte to the I/O address. In examining the program listed in Figure 2.17, note that if a valid LED number in the range from 0 to 7 is not input in step 4, all of the LEDs on the CMS I/O board will light, because in line 100 the data was set to 255 and will not be changed if there is an error. This will be an error indication. As an alternative, we could change the program to look for this error condition in software. To do this, we would simply write another loop to check for this condition. A program to realize the flowchart of Figure 2.16 is presented in Figure 2.17.

```
 10 REM THIS PROGRAM WILL OUTPUT A BYTE TO THE CMS I/O BOARD
 20 REM THAT IS INSTALLED IN A VIC-20 COMPUTER.
 50 REM START OF THE PROGRAM
 60 PRINT "WHAT I/O ADDRESS? ";
 70 INPUT S
 80 PRINT "WHICH LED ON THE CMS I/O BOARD DO YOU WANT LIT? ";
 90 INPUT L
100 L1 = 255
110 IF L = 0 THEN L1 = 1
120 IF L = 1 THEN L1 = 2
130 IF L = 2 THEN L1 = 4
140 IF L = 3 THEN L1 = 8
150 IF L = 4 THEN L1 = 16
160 IF L = 5 THEN L1 = 32
170 IF L = 6 THEN L1 = 64
180 IF L = 7 THEN L1 = 128
190 REM
200 POKE S,L1
210 PRINT
220 GOTO 80
```

***Figure 2.17**–BASIC program to realize the flowchart given in Figure 2.16.*

## 2.7 EXAMPLE 2: LIGHTING A COMBINATION OF LEDS

The second hands-on example will allow the user to light any combination of the LEDs on the CMS I/O board. You are asked to determine which LEDs you want lit, and then calculate the output variable weight necessary to accomplish this.

For example, suppose you wished to turn on LEDs 0, 5, and 7 on the CMS I/O board. The POKE weight to be output would be equal to the weight of D0 plus the weight of D5 plus the weight of D7. This equals D0=1 + D5=32 + D7=128 = 161. If the number 161 is output to the correct address, LEDs D0, D5 and D7 will be set to a logical 1 in the output data byte.

A flowchart for the program to be written is shown in Figure 2.18. A program to realize the flowchart is given in Figure 2.19.

With the CMS I/O system installed in the computer, try out the program given in Figure 2.19. This system is simply a means of visually verifying the calculated results. Use the LED patterns (a–g) shown and calculate the weight to be input. Verify your answer by executing the program. The answers to the patterns are given at the end of pattern (g).

a. LEDs D0, D4, D7 are lit.

b. LEDs D0, D3, D5, D6 are lit.

c. All LEDs are lit.

d. No LEDs are lit.

e. LED D6 is lit.

f. LEDs D1, D3, D5, D7 are lit.

g. LEDs D0, D2, D4, D6 are lit.

Answers to the LED patterns:

a = 145, b = 105, c = 255, d = 0, e = 64, f = 170, g = 85

## 2.8 EXAMPLE 3: A COUNTING PROGRAM

In this third hands-on example, we will write a program to light up the LEDs on the CMS I/O board in the following specified sequence.

At first, all of the LEDs will be turned off. Next, the number 1 will be output from the computer to the I/O board. This will turn on the LED that corresponds to a weight of 1. Only the LED labeled D0 will be lit.

The program will delay long enough for the user to see that the correct LED is lit. Next, the program will output the number 2 from the computer to the I/O board. This will turn on the LED that corresponds to a weight of 2. The LED labeled D1 will be lit at this time. Again, the program will delay long enough for the user to see the LED pattern on the I/O board.

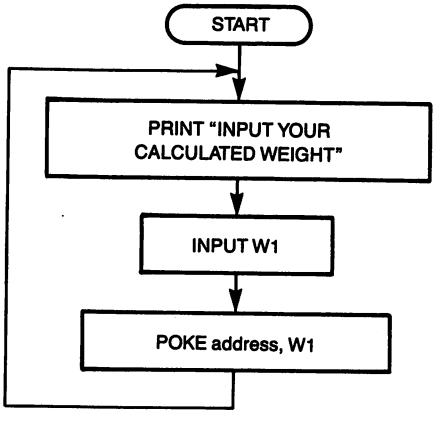



***Figure 2.18**–Flowchart for inputting the weight to POKE at the output connector.*

```
20 REM FIRST INPUT THE CALCULATED WEIGHT
30 PRINT "INPUT THE CALCULATED WEIGHT"
40 INPUT W1
50 POKE 38912,W1
60 GOTO 30
```

***Figure 2.19**–BASIC program to realize the flowchart given in Figure 2.18.*

The program now outputs the number 3. This will turn on the LEDs corresponding to a weight of 3. LEDs D0 and D1 will be lit. As before, the program will now delay to give the user time to see the LED pattern.

This process is repeated, each time incrementing by 1 the number to be output. The result will be the turning on and off of the LEDs in a manner that will show the output weight increasing by 1 for each POKE statement.

The flowchart for this program is shown in Figure 2.20. A BASIC program to realize this flowchart is given in Figure 2.21.

If you know a little more BASIC than we have tested so far, you might try the following variations. After you have had a chance to load and run the program of Figure 2.21, try to realize the program using fewer BASIC statements. Next, try speeding up and slowing down the wait time for the program. Note the effect on the output display LEDs.

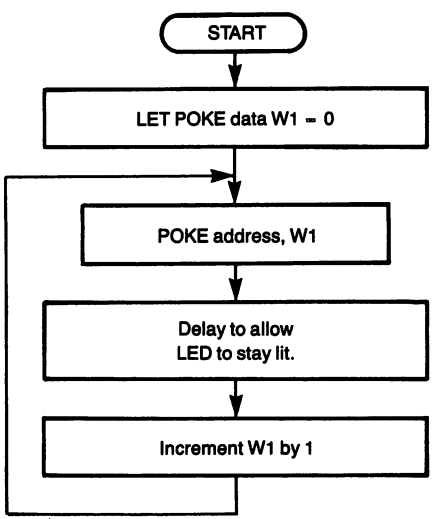



***Figure 2.20**–Flowchart for a counting program.*

```
 20  REM COUNTING PROGRAM FOR THE VIC-20 PC
 30  REM SET THE FIRST POKE DATA EQUAL TO 0
 40  W1 = 0
 50  POKE 38912,W1
 60  REM
 70  FOR I = 1 TO 1000
 80  NEXT I
 90  REM WE JUST DELAYED FOR A WHILE
100  W1 = W1 + 1
110  IF W1 > 255 THEN W1 = 0
120  GOTO 50
130  END
```

***Figure 2.21**–BASIC program to realize the flowchart of Figure 2.20.*

## 2.9 EXAMPLE 4: A TRAVELING-LIGHT PROGRAM

In our final example, a program will be written to make the LEDs turn on and off in a different sequence from that of example 3. This sequence will give the visual illusion that the light on the CMS I/O board is traveling. This is the same effect one encounters in a marquee sign, where it appears that the light is moving around the outside of the sign.

This program will be very similar to the one presented in example 3. That is, we will output a byte to the CMS I/O board and then delay for a while before outputting another. The program starts by outputting a byte that will light only LED D0. An output number of 1 will accomplish this. Next, a byte is output that will light D1. This is the number 2. The number 4 is output next, as it will light only D2. The idea is to output a number that corresponds to a single bit weight only. The remaining numbers to be output will be 8, 16, 32, 64 and 128. A flowchart for this program is shown in Figure 2.22. Figure 2.23 shows a BASIC program to realize the flowchart.

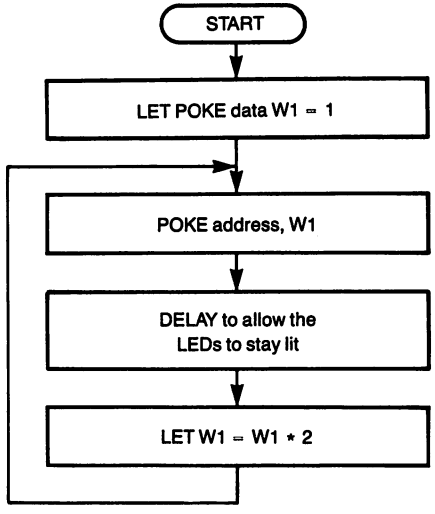



***Figure 2.22**–Flowchart for a traveling-light program.*

```
 20  REM TRAVELING LIGHT PROGRAM
 30  REM SET FIRST POKE DATA EQUAL TO 1
 40  W1 = 1
 50  POKE 38912,W1
 60  REM NOW TO DELAY
 70  FOR I = 1 TO 1000
 80  NEXT I
 90  REM NOW TO SHIFT THE DATA BIT LEFT
100  W1 = W1 * 2
110  IF W1 > 128 THEN W1 = 1
120  GOTO 50
130  END
```

***Figure 2.23**–BASIC program to realize the flowchart of Figure 2.22.*

Load the program and run it. Verify that the LEDs turn on and off in the specified sequence. After the program has run correctly, try these four variations:

a. Make the light travel backwards from the way it is now going.

b. Make the light "bounce." That is, when it gets to D7, make it travel back to D0. When it gets to D0, make it travel back to D7.

c. Make the light travel from the center LEDs to each edge. That is, start by turning on LEDs D3 and D4. Next, turn on LEDs D2 and D5. Then, turn on LEDs D1 and D6. Finally, turn on LEDs D0 and D7.

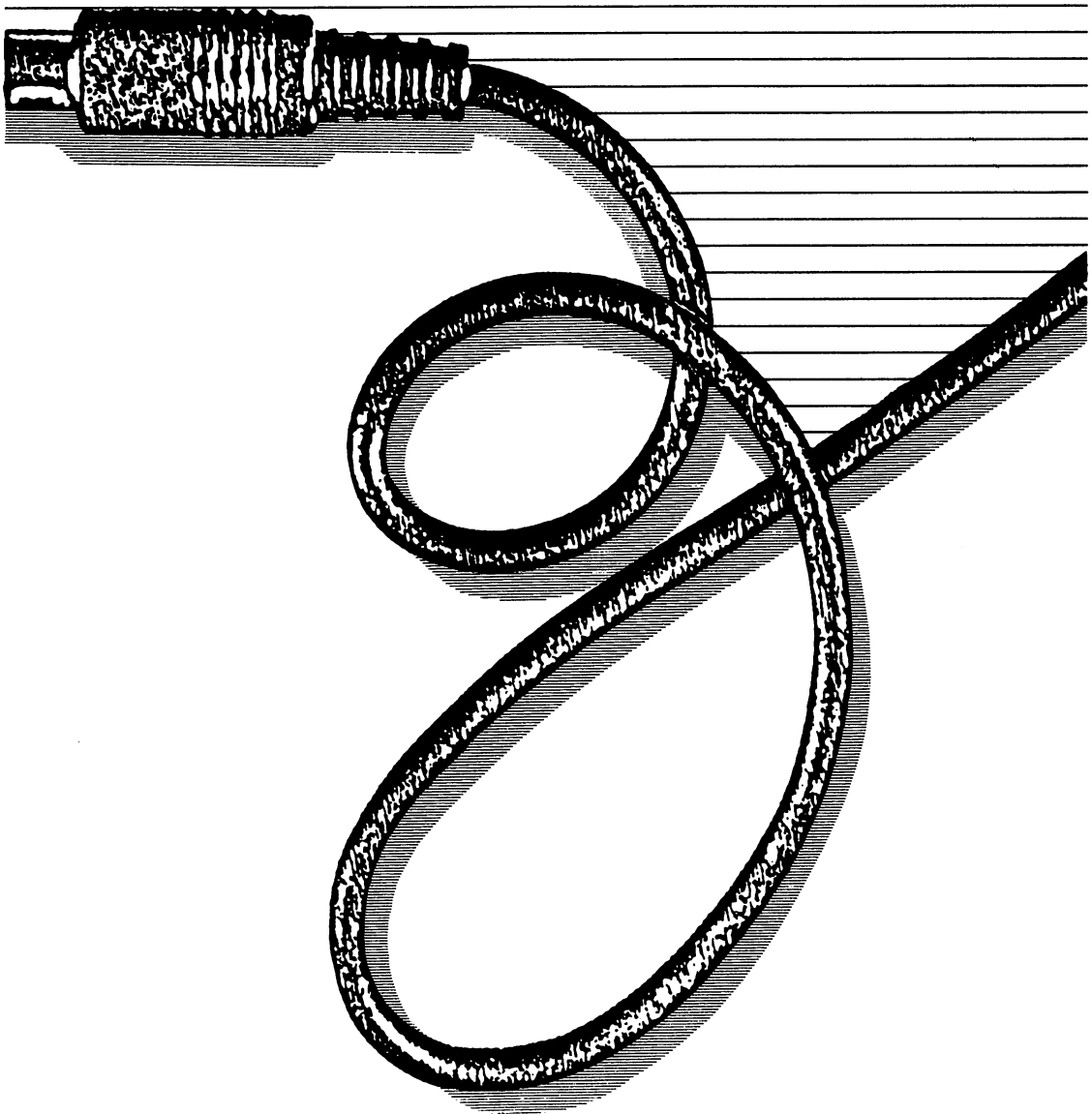
d. Make the light bounce from the center to the edges and back to the center again.

## 2.10 SUMMARY

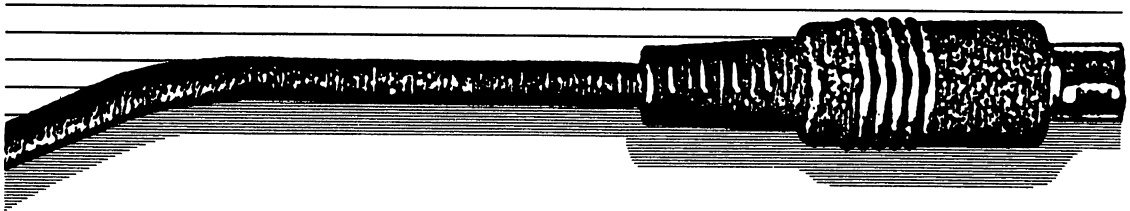
In this chapter we have covered the basics of outputting information with the VIC-20 Personal Computer. We began by plugging a simple output device into the VIC-20. The language we used for outputting was BASIC. The POKE instruction was discussed in detail, and we covered how to calculate the address needed for the POKE. Data to be output was shown, and we discussed how to set any bit in the output byte to a logical 1 or a logical 0. Finally, four hands-on examples were given to enable you to verify your understanding of the main points of outputting information with the VIC-20 computer.

When you have mastered the information presented in this chapter, half of the VIC-20 connection has been made. Chapter 3 will discuss the second half of this connection, how to input information to the VIC-20 from an external source.

# INPUTTING DATA TO THE VIC-20 COMPUTER



# 3



In this chapter we will discuss how to input digital information to the VIC-20 Personal Computer from an external device connected to the expansion socket, by means of a BASIC program. After the information is input to the program, we will show ways of interpreting it using software.

## 3.1 OVERVIEW OF INPUTTING DATA

To begin our discussion of inputting information to the VIC-20, let us review exactly what our goal is, by referring to the block diagram in Figure 3.1. We see in this diagram that an external device sends digital information (data) to the VIC-20. In order to electrically input the data, the computer must be able to accept the information and then interpret what was accepted.

As you look at Figure 3.1, certain questions may come to mind. For example, "How does the computer electrically know that the external device is ready to send information?" The answer to that question is covered in the broad topic called "handshaking." Handshaking refers to the process by which data is exchanged between a computer and an external device in an organized fashion. In general, when the external device has information or data ready to send to the

computer, another line connected to the computer will electrically indicate that the data is ready. At that time, the computer will input, or accept, the information. If the computer is outputting data, another "handshake" line will electrically inform the external device that data is ready to send.

There are other types of handshaking that may be performed, including *interrupts* and *Direct Memory Access (DMA)*. Those types of handshaking systems are beyond the scope of this book, but they are discussed elsewhere in the literature. However, in Chapter 8 we discuss a handshaking system in detail. For our purposes here, we can assume that the data from the external device is always present when the computer requests it.

Using this assumption, we can concentrate our efforts on how to transfer the information into the VIC-20, using a BASIC program. In review, our task in this chapter will be to understand how data is input from an external source to a program running in BASIC on the VIC-20. Once the data is input, ways of processing it with software to make logical decisions will be shown.

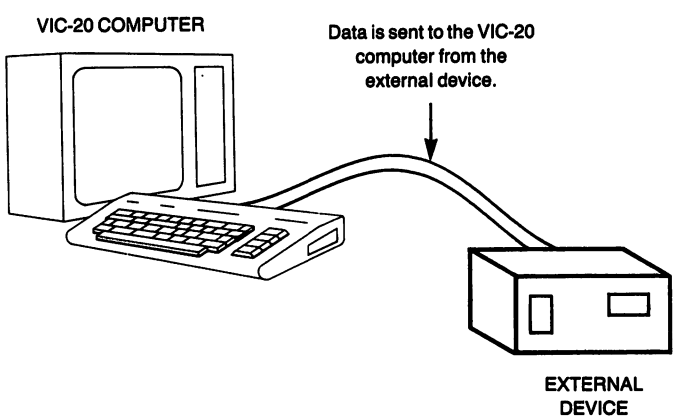



***Figure 3.1**–The object of inputting data is to send information from an external device to the VIC-20 computer. The information must be in an electrical form the computer will understand.*

## 3.2 THE CMS INPUT BOARD FOR THE VIC-20 PERSONAL COMPUTER

The physical connection used for inputting the electrical data to the VIC-20 computer will be made via the CMS I/O system. We became familiar with this I/O board in Chapter 2, when we discussed the mechanics of outputting data from the VIC-20. Besides functioning as an output device, the CMS I/O system is also electrically capable of inputting data to the computer. Figure 3.2 shows a pictorial diagram of the input section of the CMS I/O board hardware, located on board VIC-202. The I/O system should be installed in the VIC-20 exactly as described in Section 2.1 of Chapter 2. The remainder of this discussion will assume the reader has installed the CMS I/O system correctly.

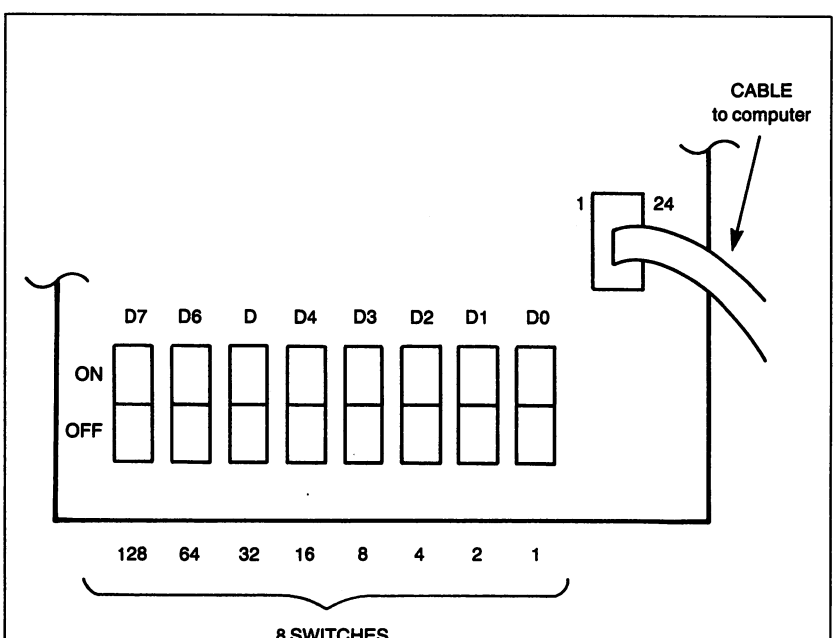



***Figure 3.2**–Pictorial diagram showing the physical switches on the CMS I/O system input section. Each switch is marked with a "D" label and its corresponding weight, and is either "ON" or "OFF."*

Notice, in Figure 3.2, that there are eight switches on the CMS I/O board input section. These switches will be set to an OFF or ON position. When the switch is OFF, it corresponds to a setting of logical 0 on a particular input line. When the switch is ON, it corresponds to a setting of logical 1 on a particular input line.

Each of the switches in Figure 3.2 is assigned a unique label: D0, D1, D2, D3, D4, D5, D6, and D7. The label corresponds to the physical signal line that will be input to the VIC-20.

It is important to know that the switches in the CMS I/O system input section operate in parallel. That is, *any signal line, D0 through D7, can be set to a logical 1 or a logical 0. Each line is set independently of any other signal line.*

## 3.3 INPUT SOFTWARE

The CMS I/O system will allow us to physically set the input lines to any desired logical state. Now we will turn our attention to the problem of how the programmer can electrically request information from the external device. The instruction used to input information is PEEK. Like the POKE instruction we used in Chapter 2, PEEK is explained fully in your user's manual and elsewhere, but we will briefly summarize its operation here. A PEEK instruction will appear in a BASIC program like this:

```
A1 = PEEK(address)
```

Let us discuss exactly what each part of this instruction does. The variable name "A1" could be any BASIC numeric variable. We used "A1" here only as an arbitrary example; it could have been replaced by T1, Z5, C(3) or any valid BASIC numeric variable name. The important point is that when information is input using the PEEK instruction, variable A1 will be set equal to it.

The (address) part of the PEEK instruction will electrically inform the VIC-20 PC which I/O address the data will be input from. The definition of this address is exactly the same as the address definition used for the POKE instruction, which we discussed in Chapter 2. Remember from that discussion that we can input information from any I/O circuit simply by using the correct I/O address. The CMS I/O system will have an address of 38912. In a BASIC program, an instruction that

would allow you to input information from the CMS system would appear as:

```
A1 = PEEK(38912)
```

After this instruction is executed, the variable A1 will be equal to the information that was input from address 38912.

Let us take another example. Suppose the I/O slot address is a variable itself. That is, the BASIC program will prompt the user and ask for the I/O address. The I/O address will be stored in a BASIC variable. In this example, we will assume that the I/O address was stored in the variable S3. The form of the PEEK instruction would be:

```
A1 = PEEK(S3)
```

This will set A1 equal to the numerical value of the information read from the I/O address S3. Figure 3.3 shows a BASIC program that will operate in the way described for this example.

```
10 PRINT "INPUT THE I/O ADDRESS FOR COMMUNICATION";
20 INPUT S3
30 A1 = PEEK(S3)
40 END
```

***Figure 3.3–*** *BASIC program to ask the user which I/O address to input data from.*

## 3.4 INTERPRETING THE INPUT INFORMATION

Up to this point in our discussion of inputting data, we have shown the software required to get the input information into a BASIC program, where it will reside in a valid BASIC variable. This section will focus on ways to interpret the information that was input to the BASIC variable. During this discussion we will be building on the information that was presented in Section 2.4 of Chapter 2.

When the VIC-20 inputs data from an I/O address, it is electrically inputting the logical voltage levels of eight separate signal lines. These signal lines are labeled D0, D1, D2, D3, D4, D5, D6, and D7. (See Figure 3.4.)

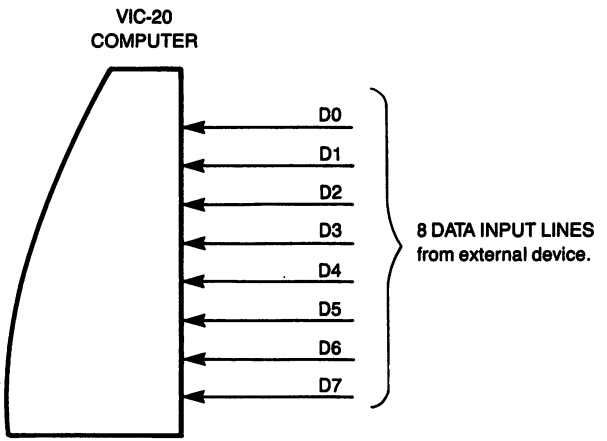



***Figure 3.4**–Eight separate signal lines are connected to the VIC-20 computer expansion connector slot in order to input external information.*

Each physical signal line is assigned a numerical weight by the computer. Since each line corresponds to a bit position, these weights are the same as those discussed in Chapter 2 and listed in Figure 2.11. (Again, it should be remembered that these weights are not arbitrarily assigned, but correspond to powers of two; that is, to binary numbers. A few minutes spent familiarizing yourself with the binary numbering system will be of great value in understanding how your computer works.) When an input signal line is a logical 1, its corresponding weight is summed. When an input signal line is a logical 0, its corresponding weight is not summed. The resulting sum is the value that will be stored into the BASIC variable used in the PEEK instruction.

For example, suppose the external hardware was connected to I/O address 38956. We would use this address in the PEEK instruction. It is further assumed that the external hardware is sending data that has lines D0, D4, D5, and D7 set to a logical 1. The remaining lines, D1, D2, D3, and D6, are set to a logical 0. The external input lines are set by the device sending the data to the computer. (See Figure 3.5.)

When we wish to obtain the information from the external device used in this example, the PEEK instruction will appear as:

```
A1 = PEEK (38956)
```

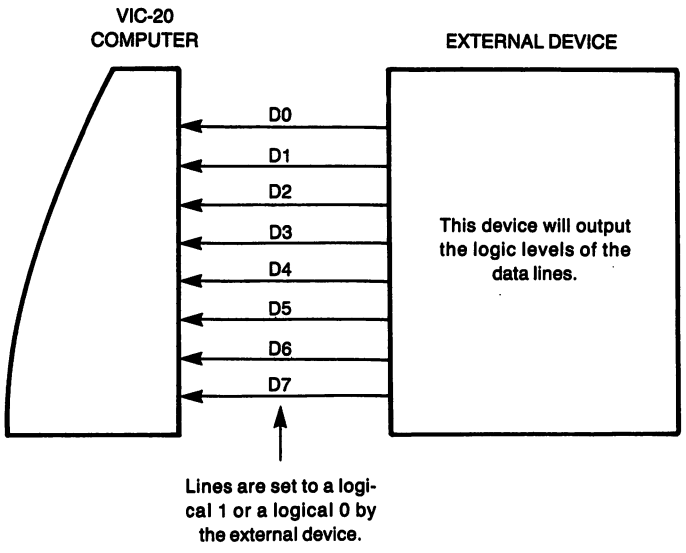



***Figure 3.5**–The external input device controls the logical voltage levels of the input lines to the VIC-20 computer.*

After this instruction is executed, the BASIC variable A1 will be equal to the summation of the weights of all the data input lines at I/O address 38956. (Notice that this address is not the one we have used previously; however, it is within the range of addresses assigned to the expansion connector, 38912–39935.) The summation will include the weights of all the input lines that were set to a logical 1 during the execution of the PEEK instruction.

In our example, the weights that will be summed are D7 = 128, D5 = 32, D4 = 16 and D0 = 1. The resulting summation would yield 128 + 32 + 16 + 1 = 177. The value of the variable A1 would be equal to 177 after the execution of the PEEK instruction.

We can expect the variable used in the PEEK instruction to be greater than or equal to 0 and less than or equal to 255. A value of 0 is returned when no input lines are set to a logical 1 at the I/O address, and a value of 255 is returned when all of the input lines are set to a logical 1 at the I/O address.

To further illustrate this point, let us consider another example. In this example it is assumed that the input device is setting data lines D6, D5, D4 and D1 to a logical 1. All other data input lines are set to a logical 0. (See Figure 3.6.) The input device is physically connected to I/O address 38960. Input information is read using the PEEK instruction. The BASIC instruction for this example will appear as:

```
R = PEEK(38960)
```

What will be the value of R after the PEEK instruction is executed? Recall that R will be equal to the sum of the weights of all input lines set to a logical 1 during the execution of the PEEK instruction. These weights will be as follows: D6 = 64, D5 = 32, D4 = 16, and D1 = 2. The resulting sum would be 64 + 32 + 16 + 2 = 114. Variable R would equal 114 after execution of the PEEK instruction. If we were to print the value of R at this time, the number 114 would appear. By using the PEEK instruction, we now have a means of inputting data from an external device. Furthermore, the data is stored in a BASIC variable. By using software, we can then operate on the variable in exactly the same way as any other BASIC variable.

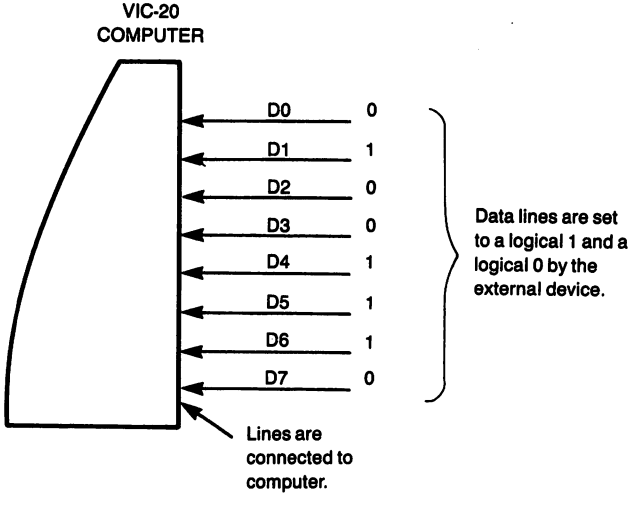



***Figure 3.6**–External data-input lines D1, D4, D5, and D6 are set to a logical 1 by the external device.*

## 3.5 CALCULATING THE BITS FROM THE INPUT VARIABLE

In the preceding section we discussed how to calculate the value of the input variable used in the PEEK instruction. That calculation was based on the assumption that we knew the logical level of the data input lines at the selected I/O address. However, in many computer control applications, we do not know which data input lines were a logical 1 and which were a logical 0 at the time of the PEEK instruction, and we need to know. The reason we need to know the logical state of the input lines is that each line input by the external device can mean something different. For example, D0 might logically inform the user that the temperature is too high, D1 might be used to indicate that some lights in a home were left on, and so on. Therefore, we need to examine the logical condition (1 or 0) of *each* data line input during the PEEK instruction. This can be accomplished using software.

There are several ways of doing this in a BASIC program; we will present only one method here. This technique is designed to help you understand exactly what is needed and what is occurring, rather than to operate as efficiently as possible. However, as a BASIC program, it *does* work. As you become more familiar with this type of programming, new and more efficient ways of performing the transformation and testing will become apparent.

Here is one way to do it. We can start by applying our knowledge of how the PEEK variable was originally formed to the problem of "deciphering" it. Recall that the variable was formed by summing the weights of the data input lines that were a logical 1 during execution of the PEEK instruction. What we will do is discover, by a process of subtraction, which individual weights were used to obtain the sum. Once these weights are known, the corresponding input lines that were set to a logical 1 are also known. All other input lines must have been a logical 0 during the execution of the PEEK instruction.

For example, suppose we executed this PEEK instruction:

```
A = PEEK(38912)
```

After this instruction was executed, the variable A would have a value between 0 and 255, inclusive. The value would depend on which of the eight data input lines, D0–D7, were set to a logical 1. For purposes of illustration, we can assume that the variable A was equal to

183 after the PEEK instruction. At the outset, we can see that at least one of the data input lines was set to a logical 1, because the value of A is not 0. We also know that at least one of the data input lines is a logical 0, because the value of A is not 255. But we do not know which input lines were set to a logical 1 and which input lines were set to a logical 0 during the PEEK instruction. That is precisely the problem we are going to solve.

Our job now is to determine which of the data input lines was a logical 1 during the input instruction. This can be done in the following way. Subtract from the variable A the weight of each input line, starting with the weight of D7. If the result of the subtraction is less than 0, we know the weight subtracted was not used to obtain the sum. Let's go through some examples to show exactly what is meant.

Suppose a variable returned from the PEEK instruction was equal to 125. Using our plan, the weight of D7 is subtracted from 125. This would yield a subtraction of 125 − 128, which is less than 0. Therefore, we know that the weight of D7 was not used to obtain the original sum, and that D7 is a logical 0.

We then proceed with the next weight in line, the weight of D6. The weight of D6 is subtracted from 125. The result would be 125 − 64 = 61. The result of this subtraction is not less than 0. Therefore, the weight of D6 was used in the original summation, and D6 is a logical 1.

When we find that a weight was used in the summation, its value is subtracted from the starting number and the resulting value is tested against the next weight in the line. In this case the next weight in the line after D6 is D5. The weight of D5 is subtracted from the new value, 61, not the original value, 125. This will give 61 − 32 = 29. This value is not less than 0. Therefore, the weight of D5 was used to obtain the original sum, and D5 is also a logical 1.

At this time we know that the weights of D6 and D5 were used in obtaining the original sum of 125. Proceeding further, we test the next weight, D4, against the new value, 29. (The value 29 is equal to the original number, 125, minus the known weights, D6 and D5.) Subtracting 16 from 29, we get a result of 13. This result is not less than 0. Therefore, the weight of D4 was used in obtaining the original sum. We continue the process by subtracting the next weight from the new value, 13.

The next weight in line is D3. The subtraction 13 − 8 = 5 yields a

number greater than 0. From this we know that the weight of D3 was used in obtaining the original sum. The next weight, D2, is tested against the new value, 5.

The resulting subtraction, 5 − 4 = 1, gives a number that is not less than 0. Again, we know that the weight of D2 was used to obtain the original sum. At this point we know that the weights of D6, D5, D4, D3, and D2 were used to obtain the original sum.

We next test against the weight of D1. This subtraction would yield 1 − 2 = −1. This result is less than 0. Because of this, we know that the weight of D1 was not used to obtain the original sum. Finally, we proceed to test against the weight of D0.

Since the subtraction of the weight of D1 yielded a negative result, the testing value, 1, is not changed. Subtracting the weight of D0 would give 1 − 1 = 0. This result is not less than 0, so we know that D0 was used to obtain the original sum of 125. At the conclusion of this process, we know that the weights of the data lines D6, D5, D4, D3, D2, and D0 were used to obtain the original sum. Therefore, these input data lines were a logical 1 during the execution of the PEEK instruction. Further, we know that the data input lines D7 and D1 were a logical 0 during the execution of the PEEK instruction.

The procedure described above is, essentially, how the weights of the data lines are tested. If at any point in the testing the result is 0, then we can stop and test no further, because all the remaining data lines must be a logical 0.

To further illustrate this procedure, let us consider another example, detailing the steps in outline form. We will assume that the variable returned from the PEEK instruction this time was 183. We need to know which data lines were a logical 1 and which were a logical 0. The procedure is outlined below:

1. Subtract the weight of D7 from the original variable, 183. This gives

   **183 − 128 = 55**

   The result is greater than 0. Therefore, the bit weight of D7 was used to obtain the original sum. D7 was a logical 1 during the PEEK instruction. We set the variable A2 equal to 55.

2. We now subtract the bit weight of D6 from the new value of variable A2.

   **55 − 64 = −9**

The result is less than 0. This tells us that the bit weight of D6 was not used to obtain the original sum of 183, and that D6 was a logical 0 during the PEEK instruction.

3. We next subtract the weight of D5 from the variable A2. Note that A2 was not changed in step 2, because the result of the subtraction was less than 0.

   ```
   55 - 32 = 23
   ```

   Since the result of the subtraction is not less than 0, we know that this weight was used to obtain the original sum. Therefore, D5 was a logical 1 during the PEEK instruction. The variable A2 is now set to 23, because the result of the subtraction was not negative. Notice that as a weight is used to obtain the sum, it is subtracted from the current value of the variable A2.

4. Next we subtract the bit weight of D4 from the variable A2.

   ```
   23 - 16 = 7
   ```

   The result is not less than 0. Therefore, the bit weight of D4 was used to obtain the original sum. Data input line D4 was a logical 1 during the PEEK instruction. The variable A2 is now equal to 7.

   Let us stop at this point to examine what information we have concerning the input data lines. From steps 1–4, we know that bit D7 = 1, D6 = 0, D5 = 1, and D4 = 1. The checking is resumed starting with the weight of D3.

5. We subtract the bit weight of D3 from the new variable A2.

   ```
   7 - 8 = -1
   ```

   The result is a number less than 0. Therefore, we know that the weight of D3 was not used in obtaining the original sum of 183. This means D3 was a logical 0 during the PEEK instruction.

6. Testing the next bit weight, D2, we obtain the result:

   ```
   7 - 4 = 3
   ```

   The result is a number that is not less than 0. Therefore, input line D2 was a logical 1 during the PEEK instruction. The variable A2 becomes 3.

7. Now test bit D1. The result of this subtraction is

   3 - 2 = 1

   We see that the D1 input line was a logical 1 during the PEEK instruction.

8. The final bit to test is D0. The result of this test is:

   1 - 1 = 0

   Since the result of the subtraction was not less than 0, we know that the weight of D0 was used to obtain the original sum.

The results of the bit testing can be summarized thus:

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | *bit labels* |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | *logical values* |

These results are easy to check, as we can simply sum the weights of all the bits that are a logical 1. The result of this summation should be the number we started with, 183.

```
  128 = D7
+  32 = D5
+  16 = D4
+   4 = D2
+   2 = D1
+   1 = D0
-------
  183      The result checks.
```

When we read through the steps required to make these tests by hand, the procedure seems quite tedious. Fortunately, we have the computer to make these checks for us. Since we know what tasks the computer has to perform, the first step in writing a computer program to perform these checks is to design a task flowchart outlining the steps of the procedure. Such a flowchart is shown in Figure 3.7.

A BASIC program to realize this flowchart on the VIC-20 computer is given in Figure 3.8. A second example of a BASIC program that will implement the flowchart of Figure 3.7 is given in Figure 3.9. The difference between the two programs is in the second program's use of FOR/NEXT loops to test the individual weights. In Figure 3.8 each weight is tested in a sequential fashion. Figure 3.9, on the other hand, tests each weight by reducing the variable A3 by half at each pass

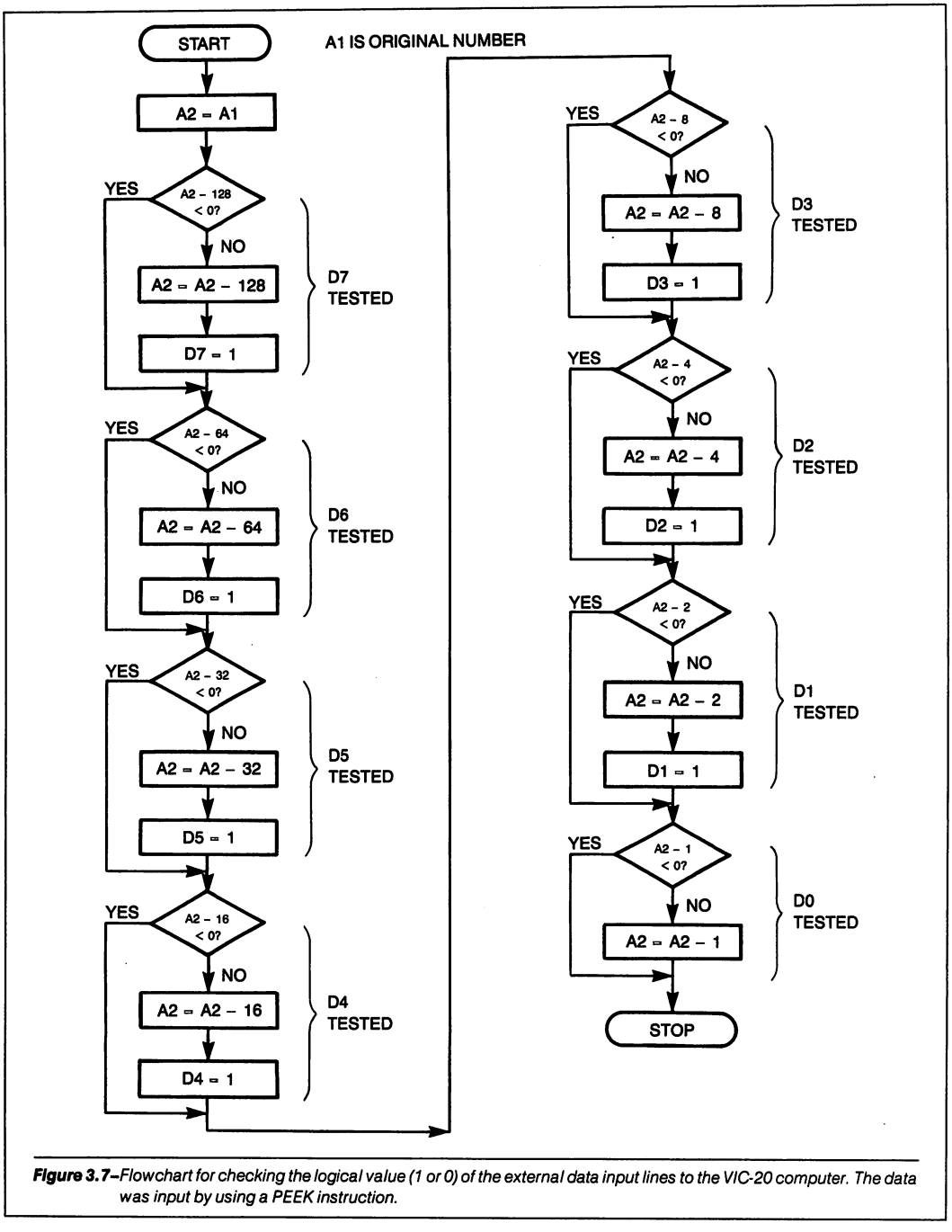



***Figure 3.7**–Flowchart for checking the logical value (1 or 0) of the external data input lines to the VIC-20 computer. The data was input by using a PEEK instruction.*

through the loop. In Figure 3.8 we simply wrote a statement that set up the new test value.

Using the testing technique described above, we will be able to tell exactly which data input lines were a logical 1 and which data input lines were a logical 0 during the execution of a POKE instruction. We are now ready to practice inputting data to the VIC-20.

```
 10 PRINT "INPUT THE ORIGINAL NUMBER BETWEEN 0 - 255 ";
 20 INPUT A1
 30 A2 = A1
 35 REM SET ALL D VALUES EQUAL TO ZERO (LINES 40-110)
 40 D0 = 0
 50 D1 = 0
 60 D2 = 0
 70 D3 = 0
 80 D4 = 0
 90 D5 = 0
100 D6 = 0
110 D7 = 0
195 REM D7 TESTED (LINES 200-220)
200 IF A2 - 128 < 0 THEN 230
210 A2 = A2 - 128
220 D7 = 1
225 REM D6 TESTED (LINES 230-250)
230 IF A2 - 64 < 0 THEN 260
240 A2 = A2 - 64
250 D6 = 1
255 REM D5 TESTED (LINES 260-280)
260 IF A2 - 32 < 0 THEN 290
270 A2 = A2 - 32
280 D5 = 1
285 REM D4 TESTED (LINES 290-310)
290 IF A2 - 16 < 0 THEN 320
300 A2 = A2 - 16
310 D4 = 1
315 REM D3 TESTED (LINES 320-340)
```

***Figure 3.8**–BASIC program to realize the flowchart of Figure 3.7 (continues).*

```
320 IF A2 - 8 < 0 THEN 350
330 A2 = A2 - 8
340 D3 = 1
345 REM D2 TESTED (LINES 350-370)
350 IF A2 - 4 < 0 THEN 380
360 A2 = A2 - 4
370 D2 = 1
375 REM D1 TESTED (LINES 380-400)
380 IF A2 - 2 < 0 THEN 410
390 A2 = A2 - 2
400 D1 = 1
405 REM D0 TESTED (LINES 410-420)
410 IF A2 - 1 < 0 THEN 430
420 D0 = 1
430 PRINT "ORIGINAL NUMBER WAS ";A1
440 PRINT
450 PRINT "BINARY VALUE IS ";D7;D6;D5;D4;D3;D2;D1;D0
460 END
```

***Figure 3.8 (continued).***

```
10 DIM D(8)
20 PRINT "INPUT THE ORIGINAL NUMBER BETWEEN 0 AND
   255";
30 INPUT A1
35 REM SET ALL D VALUES EQUAL TO ZERO (LINES 40-60)
40 FOR I = 0 TO 7
50 D(I) = 0
60 NEXT I
70 A2 = A1
80 A3 = 128
85 REM LOOP TO TEST ALL D VALUES (LINES 90-140)
90 FOR I = 7 TO 0 STEP -1
```

***Figure 3.9**–A shorter BASIC program to realize the flowchart of Figure 3.7. In lines 80 – 130, the variable A3 is the weight of each data line. A3 will start at 128, which is the weight of D7. This variable is then reduced by half in each pass through the FOR/NEXT loop (continues).*

```
100    IF A2 - A3 < 0 THEN 130
110    A2 = A2 - A3
120    D(I) = 1
130    A3 = A3/2
140  NEXT I
150  PRINT "ORIGINAL NUMBER WAS ";A1
160  PRINT
170  PRINT "THE BINARY VALUE IS ";
180  FOR I = 7 TO 0 STEP - 1
190    PRINT D(I)
200  NEXT I
210  PRINT
220  END
```

*Figure 3.9 (continued).*

## 3.6 HANDS-ON EXAMPLE 1: CALCULATING THE WEIGHT OF THE INPUT WORD

In this first hands-on example of inputting data to the VIC-20 computer from the CMS I/O board, we will calculate the value of the input weights by hand. The input weight will be calculated according to the method described in this chapter. After calculating manually the weight, you should set the switches on the CMS I/O board to the correct value and run the program.

The program will print out the weight calculated by the computer. You can then verify whether your manual calculation was correct. Let us go over an example to see how this works. First, set the input switches on the CMS I/O board so that D0 and D4 are a logical 1. All other switches are set to a logical 0. Remember that when the switch is in the OFF position it is a logical 0, and when it is in the ON position it is a logical 1.

Now calculate the input weight the VIC-20 will see when this data is read. In this example the computer will electrically input D4 and D0 as a logical 1. This will correspond to a weight of (D4 = 16) + (D0 = 1) = (16 + 1) = 17.

After the computer has read this data it will print out the number 17. At this point we can check our calculated results against the actual computer results. In this way you can quickly verify that you understand how the input lines are weighted.

Try the program shown in Figure 3.10, using the different switch settings, a–f. The switches will be set to a logical 0. The correct weight for each switch setting is given after setting (f).

a. D1 and D7 are set to 1

b. D0 and D5 are set to 1

c. D4, D6, and D7 are set to 1

d. D1, D2, D3, D4, D6, D7 are set to 1

e. All switches are set to a logical 1

f. All switches are set to a logical 0

Answers: a = 130, b = 33, c = 208, d = 222, e = 255, f = 0.

```
10 PRINT "INPUT THE I/O ADDRESS FOR COMMUNICATION";
20 INPUT S3
30 A1 = PEEK(S3)
40 PRINT "THE DATA READ FROM I/O ADDRESS # ";S3;" = ";A1
50 END
```

***Figure 3.10**–A BASIC program to read the input data from the requested I/O address and then print the data on the screen of the VIC-20.*

## 3.7 EXAMPLE 2: READ A BYTE AND DETERMINE WHICH BITS WERE A LOGICAL 1

In this example, the computer will read an input word from the CMS I/O board and print out which data lines were equal to a logical 1 and which data lines were equal to a logical 0. This is exactly the same type of operation we discussed in Section 3.5. A general program to perform this function with the VIC-20 PC is shown in Figure 3.11. Run this program and verify that the computer will print out the correct data lines that you have set on the CMS I/O board switches. Use the switch settings given in Section 3.6.

```
10 PRINT "INPUT THE I/O ADDRESS FOR THE CMS BOARD ";
20 INPUT S3
30 A1 = PEEK(S3)
40 A2 = A1
50 REM INITIALIZE THE VARIABLES
60 D0 = 0:D1 = 0:D2 = 0:D3 = 0:D4 = 0:D5 = 0:D6 = 0:D7 = 0
70 IF A2 - 128 < 0 THEN 100
80 A2 = A2 - 128
90 D7 = 1
100 IF A2 - 64 < 0 THEN 130
110 A2 = A2 - 64
120 D6 = 1
130 IF A2 - 32 < 0 THEN 160
140 A2 = A2 - 32
150 D5 = 1
160 IF A2 - 16 < 0 THEN 190
170 A2 = A2 - 16
180 D4 = 1
190 IF A2 - 8 < 0 THEN 220
200 A2 = A2 - 8
210 D3 = 1
220 IF A2 - 4 < 0 THEN 250
230 A2 = A2 - 4
240 D2 = 1
250 IF A2 - 2 < 0 THEN 280
260 A2 = A2 - 2
270 D1 = 1
280 IF A2 - 1 < 0 THEN 300
290 D0 = 1
300 PRINT
310 PRINT "THE BINARY VALUES OF THE INPUT WERE "
320 PRINT
330 PRINT "D7 = ";D7
```

***Figure 3.11**–A BASIC program to read the input data and then print out the logical value of each data line, D0 – D7 (continues).*

```
340 PRINT "D6  = ";D6
350 PRINT "D5  = ";D5
360 PRINT "D4  = ";D4
370 PRINT "D3  = ";D3
380 PRINT "D2  = ";D2
390 PRINT "D1  = ";D1
400 PRINT "D0  = ";D0
410 END
```

*Figure 3.11 (continued).*

After you have run the program and verified that it works, make certain you understand each part of it. Now write a similar program using fewer BASIC statements. Use this new program to print out only the input lines that are a logical 1. For example, if switches D0 and D4 are set to a logical 1 position on the CMS I/O board, have the computer print out "D0, D4."

Now reverse the sense of the problem. Have the computer print out the input lines that are set to a logical 0 instead of a logical 1.

## 3.8 EXAMPLE 3: READ A WORD AND PERFORM AN ACTION

In this example we will input a byte from the CMS I/O board to the VIC-20 computer. If the byte we are inputting is equal to a certain value, we will perform a particular action; we will print a message. If the byte is not equal to that value, we will perform a different action; we will print a different message.

This type of example is very similar in principle to the process of monitoring an input device with the VIC-20 (or any other) computer. When the input condition is true, the computer will automatically perform some action based on this input. Of course, in a real application, the input would come from some piece of external hardware the computer was controlling or monitoring.

In our example, the inputs will be generated by the user from the CMS I/O board. When the data word that is input is equal to 12, the computer will print the message, "THE INPUT WORD IS EQUAL TO 12." If the input word is not equal to 12, then the computer will print

the message, "THE INPUT WORD IS NOT EQUAL TO 12 AT THIS TIME."

A flowchart for this problem is shown in Figure 3.12, and a BASIC program to realize the flowchart is given in Figure 3.13.

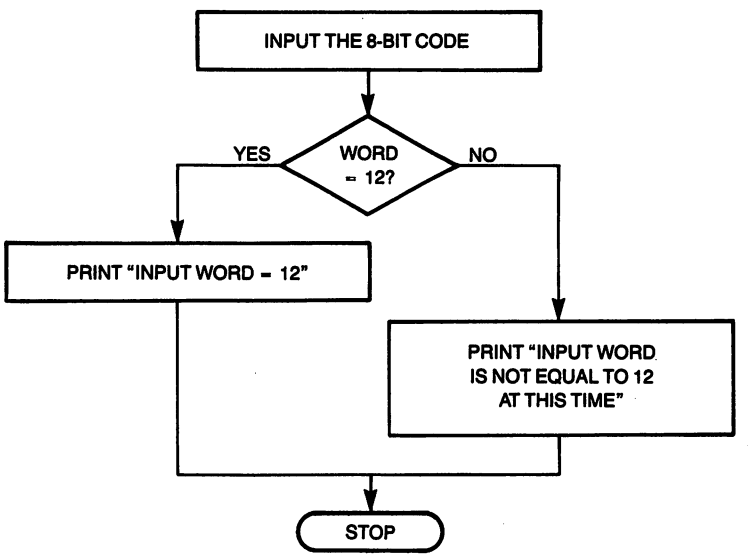



***Figure 3.12**–A task flowchart for the solution to the problem given in Section 3.8.*

```
10  PRINT "INPUT THE I/O ADDRESS FOR THE CMS BOARD ";
20  INPUT S3
30  A1 = PEEK(S3)
40  IF A1 = 12 THEN 70
50  PRINT "THE INPUT WORD IS NOT EQUAL TO 12 AT THIS TIME"
60  STOP
70  PRINT "THE INPUT WORD IS EQUAL TO 12"
80  END
```

***Figure 3.13**–A BASIC program to realize the flowchart of Figure 3.12.*

Try this program out with your VIC-20. Set the switches on the CMS I/O board to a number other than 12, and be sure that the computer will print out the error message. Now set the input word equal to 12 (that is, set D3 and D2 to a logical 1), and check to see if the computer will print out the correct message.

When you have the program running, try the following variations:

a. Have the computer check for an even number of ones in the input word. If the number of ones is even, the computer will print, "THERE ARE AN EVEN NUMBER OF ONES." If the number of ones in the input word is odd, the computer will print, "THE NUMBER OF ONES IS ODD."

b. Input a number, and have the computer subtract 25 from it; then print out the results. The output should be both your original number and the new number. Remember that it is possible to get a number that is less than 0 in this program. If the result is less than 0, print out the message, "THE RESULT WAS LESS THAN ZERO."

## 3.9 EXAMPLE 4: A COMBINATION LOCK

In this example we will make the VIC-20 Personal Computer into a combination lock. The lock will have three unique numbers that must be entered in sequence. If any of the numbers (or their sequence) is not correct, the lock will not open.

The three numbers will be 64, 128, and 7. Here is how the lock will work. A BASIC computer program will ask you to enter the first number by setting the switches corresponding to that numerical weight on the CMS I/O board. When the number is set, you will enter "L" on the computer. This signals the computer that you have entered the number and are ready for the computer to read it.

If the computer detects a wrong number, the message, "I WILL NEVER OPEN FOR YOU" will be printed. If the number is correct, the computer will print the message, "SO FAR SO GOOD." When you enter the last number correctly, the computer will print, "YOU OPENED THE LOCK." Enter an "R" to reset the combination lock. This will allow you to start over.

A flowchart of this problem is shown in Figure 3.14. Figure 3.15

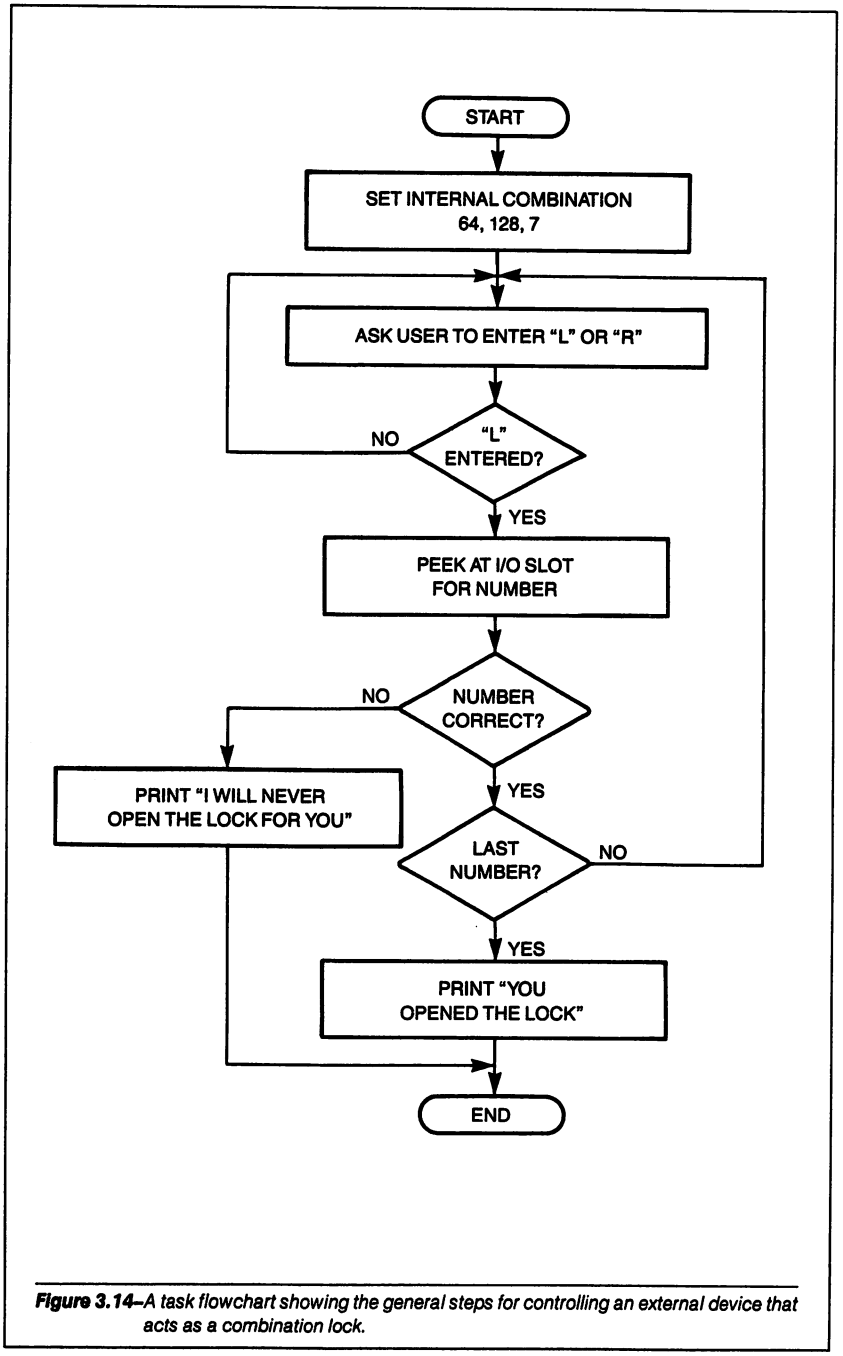



***Figure 3.14**–A task flowchart showing the general steps for controlling an external device that acts as a combination lock.*

shows one BASIC program that will realize this flowchart. Try this program out with your VIC-20 computer. Once you have this program running, try putting in the following variations:

a. Change the combination of the lock to 2, 4, 8.

b. Realize the program using fewer BASIC statements than shown in the example.

```
10  DIM N(3),L(3),A$(5)
20  REM THESE ARE THE COMBINATION NUMBERS
30  N(1) = 64
40  N(2) = 128
50  N(3) = 7
60  REM THESE ARE THE INPUT COMBINATION NUMBERS
70  L(1) = 0
80  L(2) = 0
90  L(3) = 0
100 REM START OF COMBINATION INPUTTING
110 I = 1
120 PRINT "PUT SETTING # ";I;" ON CMS INPUT SWITCHES"
130 PRINT "INPUT 'L' WHEN READY, OR 'R' TO RESET"
140 INPUT A$
150 IF A$ = "L" THEN 180
160 IF A$ = "R" THEN 70
170 GOTO 120
180 L(I) = PEEK(38912)
190 IF L(I) = N(I) THEN 220
200 PRINT "I WILL NEVER OPEN THE LOCK FOR YOU !!!"
210 GOTO 240
220 IF I = 3 THEN 260
230 PRINT "SO FAR, SO GOOD"
240 I = I + 1
250 GOTO 120
255 REM THE LOCK WILL NOW OPEN
260 PRINT "YOU OPENED THE LOCK"
270 END
```

***Figure 3.15**–A BASIC program to realize the flowchart of Figure 3.14.*

## 3.10 SUMMARY

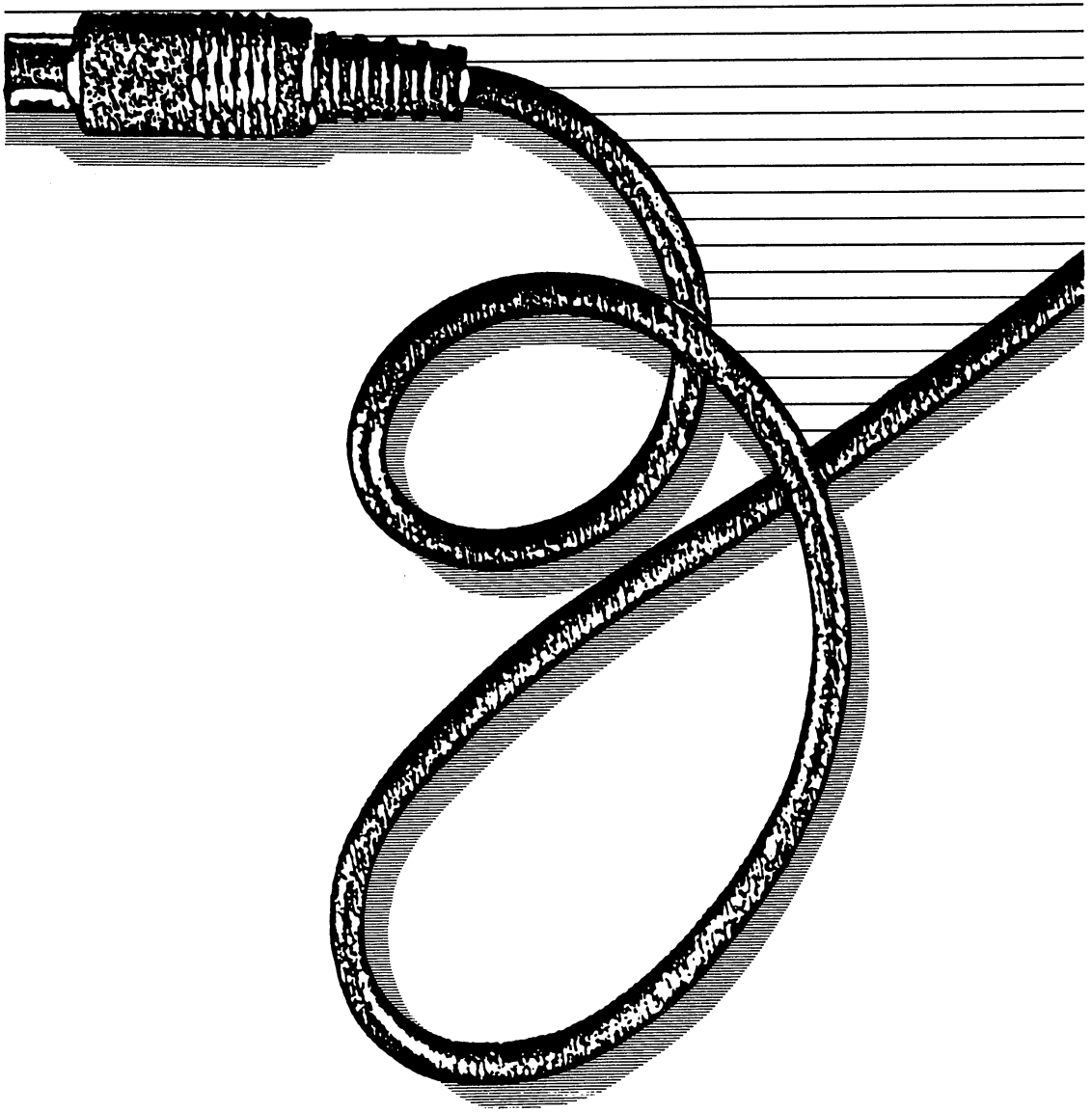
In this chapter we have discussed in detail how to input information into a BASIC program from an external device. This device will be monitored by the VIC-20 Personal Computer.

Once the information was input into the BASIC program, we discussed different methods of interpreting what the input number meant. A technique was described that will enable you to determine which input lines were a logical 1 and which input lines were a logical 0.

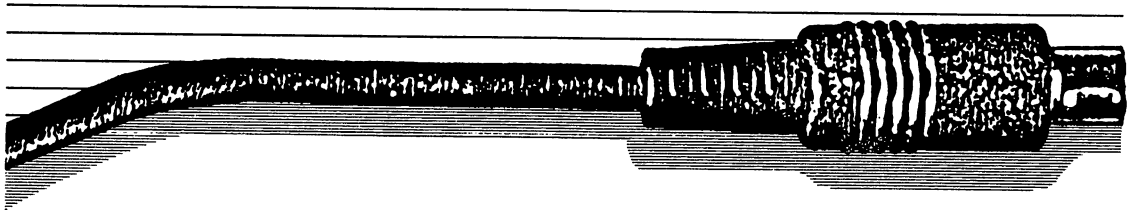
At the end of this chapter we presented some hands-on examples. These examples were designed to allow you to practice inputting information to a BASIC program, and to test your understanding of the methods outlined.

If you understand the solution to these examples, then you understand the principles of inputting data. In Chapter 4 we will present the hardware for this task, and describe how the software will interact with it. Now you are ready for the next step along the path toward making *the VIC-20 Connection.* That step is to actually connect the hardware to allow the computer to communicate with an external device.

# INPUT AND OUTPUT HARDWARE FOR THE VIC-20 COMPUTER



# 4



In this chapter we will discuss the hardware, internal and external, necessary to input and output data to and from the VIC-20. It should be noted that this discussion is meant for the beginner in computer interfacing and control. These discussions are centered around the input and output hardware architecture of the VIC-20 Personal Computer's expansion connector.

The designs shown in this chapter work, but they are not presented as the most elegant or efficient, "least-parts-count," circuits. Instead, their main function is to instruct. They are designed to enable a person who has little or no experience in digital hardware to understand the major concepts involved in the input and output of electrical information between the VIC-20 and an external device. In the chapters that follow, we will make use of the information presented here.

If this is your first exposure to any digital hardware, you may feel apprehensive about the difficulties of the subject. Furthermore, you may not *want* to learn about the hardware of the computer in any great detail. But if you can grasp the essential facts and concepts presented in this chapter, understanding how other peripheral hardware

is interfaced to the VIC-20 will be much easier. That will be true even if you never actually design a computer interface for the VIC-20 computer. Using the information given in this chapter, you will better understand other manufacturers' interfaces to the VIC-20. As we proceed through the discussion, relationships will be pointed out between the hardware described here and the software given in Chapters 2 and 3.

## 4.1 BEGINNING OUTPUT ELECTRONICS FOR THE VIC-20

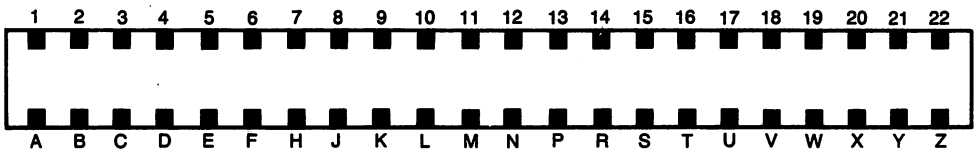
Four main digital electronic sections are of concern when discussing output hardware for the VIC-20. They are:

1. The enable circuit for the external device.
2. The READ/$\overline{\text{WRITE}}$ signal.
3. The write strobe for the circuit.
4. The output storage latches.

These four sections operate together to perform the output function. If any one of them fails to operate correctly, the output operation will not work. Let us discuss the function each section performs in an output operation. As each section is discussed, we will show how it can be realized with common digital electronic circuits for the VIC-20.

## 4.2 THE ENABLE CIRCUIT

To start, we concentrate on the function of the enable circuit in an output operation for the VIC-20. When an enable circuit is active, it means the computer will be electrically addressing the output circuit. The computer is capable of communicating with over 64,000 different circuits. Therefore, it must have some means of electrically informing any particular output circuit that it has been selected to communicate with. This communication is accomplished via the internal computer lines called *address lines.* These lines are output on the pins of the expansion I/O connector in the VIC-20 PC. Figure 4.1 shows the pinout of the expansion connector with the pin number and signal name of each pin.




| PIN # | TYPE |
|---|---|
| 1 | GND |
| 2 | CD0 |
| 3 | CD1 |
| 4 | CD2 |
| 5 | CD3 |
| 6 | CD4 |
| 7 | CD5 |
| 8 | CD6 |
| 9 | CD7 |
| 10 | $\overline{\text{BLK1}}$ |
| 11 | $\overline{\text{BLK2}}$ |

| PIN # | TYPE |
|---|---|
| 12 | $\overline{\text{BLK3}}$ |
| 13 | $\overline{\text{BLK5}}$ |
| 14 | $\overline{\text{RAM1}}$ |
| 15 | $\overline{\text{RAM2}}$ |
| 16 | $\overline{\text{RAM3}}$ |
| 17 | VR/$\overline{\text{W}}$ |
| 18 | $\overline{\text{CR/W}}$ |
| 19 | $\overline{\text{IRQ}}$ |
| 20 | NC |
| 21 | +5V |
| 22 | GND |

| PIN # | TYPE |
|---|---|
| A | GND |
| B | CA0 |
| C | CA1 |
| D | CA2 |
| E | CA3 |
| F | CA4 |
| H | CA5 |
| J | CA6 |
| K | CA7 |
| L | CA8 |
| M | CA9 |

| PIN # | TYPE |
|---|---|
| N | CA10 |
| P | CA11 |
| R | CA12 |
| S | CA13 |
| T | $\overline{\text{I/O2}}$ |
| U | $\overline{\text{I/O3}}$ |
| V | $\overline{\text{S02}}$ |
| W | $\overline{\text{NMI}}$ |
| X | $\overline{\text{RESET}}$ |
| Y | NC |
| Z | GND |

***Figure 4.1**–Pinout of the VIC-20 computer's 44-pin expansion connector. Reproduced by permission of Commodore Business Machines, Inc.*

The enable signals we are interested in are called $\overline{I/O2}$ and $\overline{I/O3}$. They are output on pins T and U of the I/O connector shown in Figure 4.1. The bar over the top of the signal name indicates that when the signal is active, doing its electrical job, its logical state is 0. At all other times this signal will be in the logical 1 state.

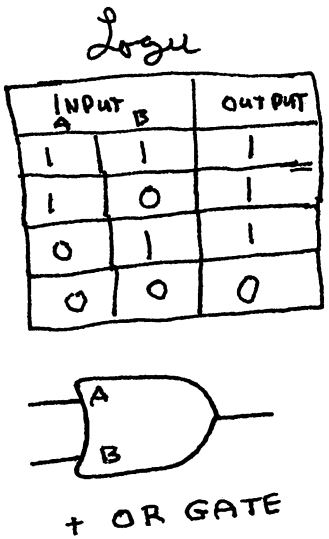
Whenever the computer selects the $\overline{I/O2}$ or the $\overline{I/O3}$ address to communicate with, the $\overline{I/O2}$ or $\overline{I/O3}$ line on the I/O expansion connector will be set to a logical 0. Either of these two select lines can be activated under software control, using the PEEK and POKE instructions. Recall that these two BASIC instructions each include an address. The logical state of the I/O output signal depends on the address specified in the PEEK or POKE instruction. For example, if we want the $\overline{I/O2}$ line on the I/O connector to go to a logical 0, we can PEEK or POKE addresses 38912 through 39935 (38912+1023). To activate the $\overline{I/O3}$ line we can specify an address from 39936 through 40959 (39936+1023). We will use the $\overline{I/O2}$ line in our examples, but from this discussion you can relate the information to the $\overline{I/O3}$ line.

The $\overline{I/O2}$ line is activated directly by the VIC-20 computer. Let us assume that our computer is operating correctly, and that the PEEK or POKE address is correct to generate the $\overline{I/O2}$ signal. Further, let us assume that this line will go to a logical 0 whenever we wish to output information from the computer to the output circuit to which the active signal is applied.

## 4.3 THE READ/$\overline{WRITE}$ (VR/$\overline{W}$) LINE

Another hardware signal that we must make use of when performing computer output is the READ/$\overline{WRITE}$ or VR/$\overline{W}$ line. This signal is output on pin 17 of the I/O connector shown in Figure 4.1. The VR/$\overline{W}$ signal is a logical 1 whenever the VIC-20 computer is reading data from the I/O slot, and a logical 0 whenever the computer is writing data to the slot. During a PEEK instruction, therefore, the VR/$\overline{W}$ line is a logical 1,because the computer is reading data. During a POKE instruction, the VR/$\overline{W}$ line is a logical 0, because the computer is writing data. The VR/$\overline{W}$ line must be electrically *qualified* in order for an output circuit to make use of it. By that we mean that the circuit must not only electrically examine the logical state of the VR/$\overline{W}$ line to

determine whether our circuit will be read or written to, but it must also examine the logical state of the $\overline{I/O2}$ line mentioned earlier. That is, the VR/$\overline{W}$ line must be "qualified" by the $\overline{I/O2}$ select line before we can use it in the external circuit. Figure 4.2 shows a block diagram of exactly what we mean.



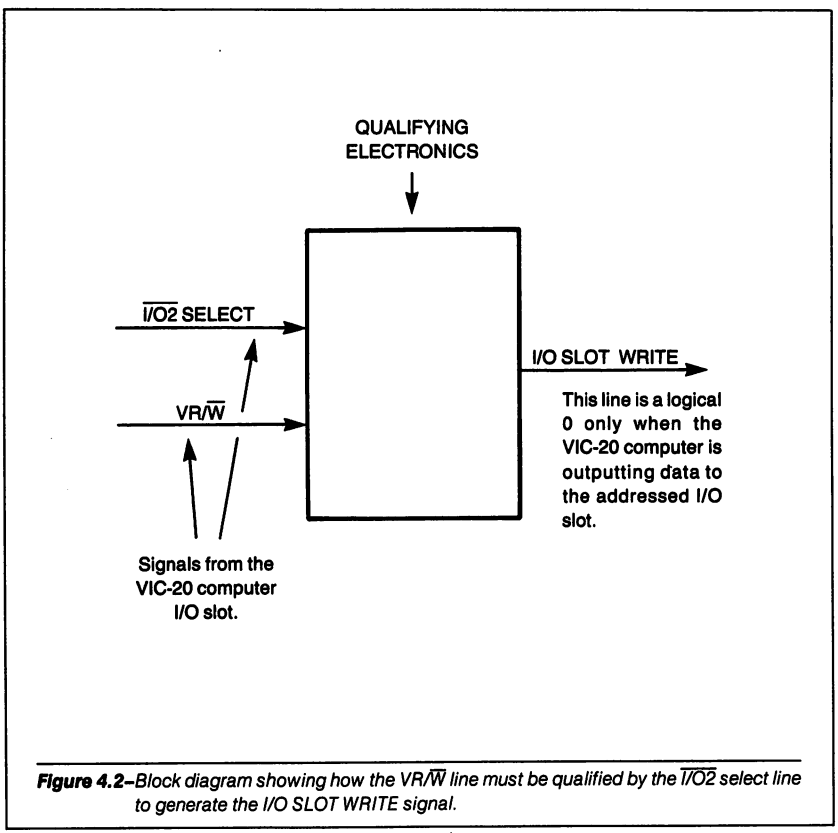




***Figure 4.2**–Block diagram showing how the VR/$\overline{W}$ line must be qualified by the $\overline{I/O2}$ select line to generate the I/O SLOT WRITE signal.*

Figure 4.3 shows a hardware realization for the qualification of the VR/$\overline{W}$ line to be used by an output circuit. In this schematic diagram we see that the $\overline{I/O2}$ signal is input to pin 1 of the 74LS32, an OR gate. The 74LS00 family of integrated circuits is widely available and will be used throughout this book. An OR gate is a digital logic device that will produce a logical 1 at its output when either input A or input B is a logical 1, or when they are both logical 1. When both inputs are a logical 0, the output is a logical 0. Input A and input B are shown as pin 1 and pin 2 in Figure 4.3. The 74LS32 integrated circuit contains

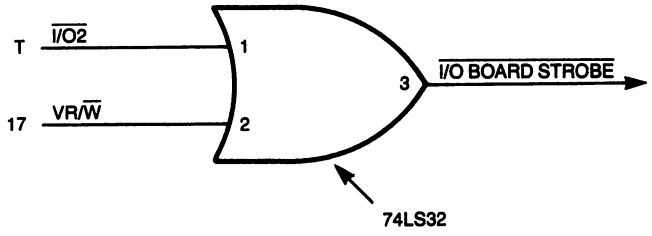



***Figure 4.3**–Schematic diagram showing how to realize the block diagram of Figure 4.2.*

four individual OR gates. The VR/$\overline{W}$ signal is input to pin 2 of the same OR gate. The output line of the OR gate, pin 3, will be a logical 0 only when both the VR/$\overline{W}$ and the $\overline{I/O2}$ line are a logical 0. That is, we have now "qualified" the VR/$\overline{W}$ line. The output of the OR gate in Figure 4.3 is given the name $\overline{\text{BOARD STROBE}}$. This line is a logical 0 only when the VIC-20 computer is outputting data to the external circuit connected to $\overline{I/O2}$.

## 4.4 THE EXTERNAL OUTPUT STROBE

The next output signal we will discuss is the signal which will strobe or write the data to the output circuit. This signal is generated by combining the $\overline{I/O2}$ and VR/$\overline{W}$ signals discussed in the previous two sections. When both of these signals are a logical 0, we generate a new signal, labeled $\overline{\text{BOARD STROBE}}$. This circuit was shown in Figure 4.3.

This strobe will become active when the computer executes the POKE instruction to the address of $\overline{I/O2}$. All timing of the computer signals is taken care of by the internal hardware of the VIC-20. All we need do is use the signals provided by the manufacturer for our circuit.

## 4.5 THE OUTPUT LATCHES

The final section of the output hardware we will discuss comprises the output latches. Output latches have the very important function of temporarily storing the data sent by the computer to an external circuit. Recall that the POKE instruction included an address and data. The address will select the I/O circuit, and the data specified will be sent to the external circuit. The output latches will store this data.

The computer will present the data to the input pins of the output latches. (See Figure 4.4.) At this time, the BOARD STROBE signal will provide a write input pulse to the latches. After the data has been written into the latches, it will remain stored there until the computer is instructed to write other data to the circuit. To put it another way, when data is output into the storage latches, it will remain there until another POKE instruction directs new data to the same address.

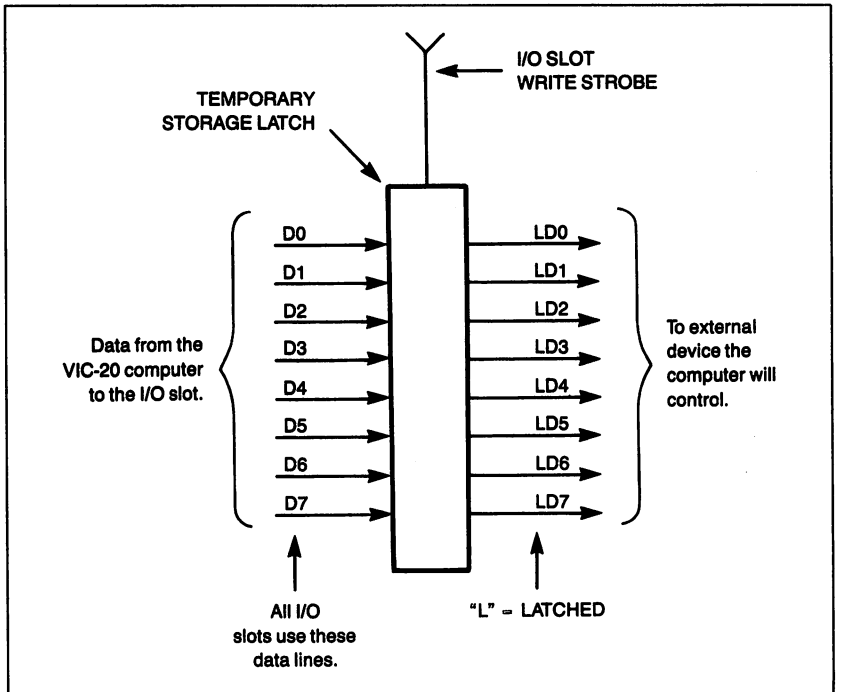



***Figure 4.4**–Block diagram showing how the data output lines from the VIC-20 computer will connect to the temporary storage latches located in the I/O expansion slot output circuit.*

The data at the output latches can now be used to control any hardware that we desire. At this point, you may not have a clear picture of exactly what hardware can be controlled. Don't worry. We will introduce some examples in later chapters. The point is that we now have a means of forcing any single logical line to switch between a logical 0 and a logical 1 level under computer control. We have achieved an important objective in the interfacing problem. When you understand how this was done, a major step has been completed.

Now let us take a closer look at the hardware of the output latch circuits. Figure 4.5 shows the latch device we will use and the pin numbers that incoming and outgoing data lines will connect to. The latch used is an 8-bit 74LS374. The VIC-20 computer will transfer eight bits of data during every output operation, so this device matches perfectly. Data from the computer is input to the latch from the VIC-20 data lines D7 – D0.

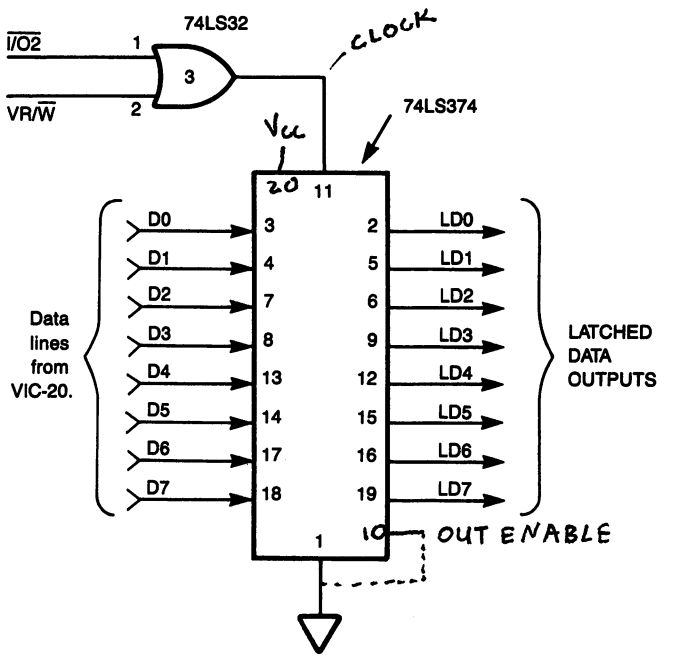



***Figure 4.5**–Schematic diagram showing the connection of the 74LS374 latch to the VIC-20 expansion connector pins.*

Data from the VIC-20 PC will be applied to the data input pins of the 74LS374. Pin 11 of the latch is the strobe input. When this line is active, it goes from a logical 0 to a logical 1. The data present at the input of the latch is "captured" and stored internally in the latch. Recall that when the strobe line is activated, it is in a logical 0 state. When this line returns to the logical 1 state, the data is latched.

We have now followed the data, along with the VR/$\overline{W}$ and $\overline{I/O2}$ from the pins of the VIC-20 expansion connector and into the latches of an I/O circuit. The data is now latched, and we are ready to use this data to control an output device.

## 4.6 THE LIGHT-EMITTING DIODES

The output device that we will be controlling in this first example will be a series of light-emitting diodes, or LEDs. Eight LEDs are used, one for each data output line. By using LEDs, we can gain experience in turning on and off selected bits of the output device. We did this in Chapters 2 and 3 by using the CMS I/O system. At this time, you may choose to construct your own I/O system using the schematics given in this chapter.

When we turn on and off selected LEDs, we are performing exactly the same function as when we control most types of external devices. We will now concentrate on how the hardware does its part of the job.

Figure 4.6 shows a schematic diagram of how an LED is electrically driven to light. We see in this diagram that the LED has approximately the same electrical symbol as a standard diode. The difference is that the LED symbol has some arrows drawn out from it, an indication that the diode is emitting light.

The LED performs approximately the same electrical function as a standard diode. That is, it will pass current when connected as shown in Figure 4.6 (forward-biased). The LED will not pass current when connected as shown in Figure 4.7 (reverse-biased). The important difference is that when the LED is forward-biased, it will emit light. The color of the light is determined by the materials used to fabricate the LED. Typical LED colors are red, green and yellow. In Figure 4.6 we see that the *anode* of the LED is connected to +5 volts. The anode is the positive (+) side of the device. The *cathode,* or negative (−) side

of the LED, is connected to one end of a resistor. To turn on the LED, the other side of the resistor must be connected to ground potential, or approximately 0.0 volts.

If you have never used an LED before, it might be fun and instructive to construct the circuit shown in Figure 4.6 and manually turn the diode on and off. This is done by connecting the anode to +5 volts and then disconnecting it. When you do this, try reversing the connection by interchanging the diode leads, and note what happens. That is, connect the cathode (−) side of the LED to +5 volts and the anode (+) side of the LED to the resistor. (See Figure 4.7.) Connect the other side of the resistor to ground again. The LED should not light under these conditions, because it is now connected in a *reverse-biased* mode.

Here is how the circuit of Figure 4.6 works. With the anode of the LED connected to +5 volts, the cathode must be connected to ground *potential* to light the diode. There must not be a direct connection of the cathode to ground. If there were, too much current would be passed through the LED, and it would burn up. Therefore, a current-limiting resistor, R1, is required. The function of R1 is to limit

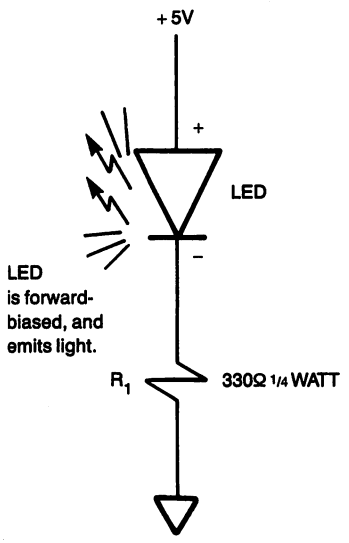



**Figure 4.6**–*Schematic diagram of a typical LED connected in the forward-biased mode.*

the current to a safe value in the LED when it is connected in the forward-biased mode. Typical LED currents are around 10 milliamperes, or .01 amperes. This amount of current will usually give a nice degree of brightness.

Multiplying a single LED by eight, we have the circuit shown in Figure 4.8. In this schematic diagram, the resistors R1 – R8 are connected at one end to the LEDs, and at the other end to the outputs of a 7406 integrated circuit. The 7406 will act as a switch to connect the resistors to ground potential. For example, when the input pin 1 of the 7406 shown in Figure 4.8 is a logical 1, the output pin 2 is connected to ground potential. Under these conditions, the LED is turned on, or forward-biased.

When the input pin 1 of the 7406 is a logical 0, the output pin 2 is not connected to ground. The output is essentially an open circuit; that is, no current is passing through it. Under this condition the LED is turned off. From these two conditions, we see that all we need to do to turn the LED on and off is to place a logical 1 or a logical 0 at the input of the 7406. As long as the input stays at a logical 1 or a logical 0, the LED will remain either on or off.

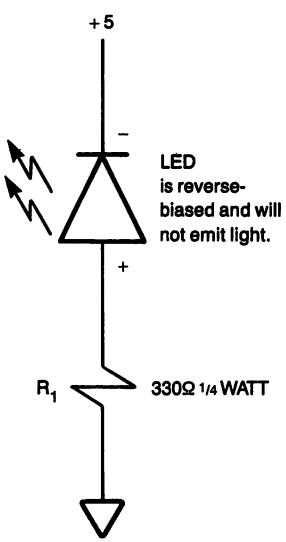



***Figure 4.7**–Schematic diagram of an LED connected in the reverse-biased mode.*

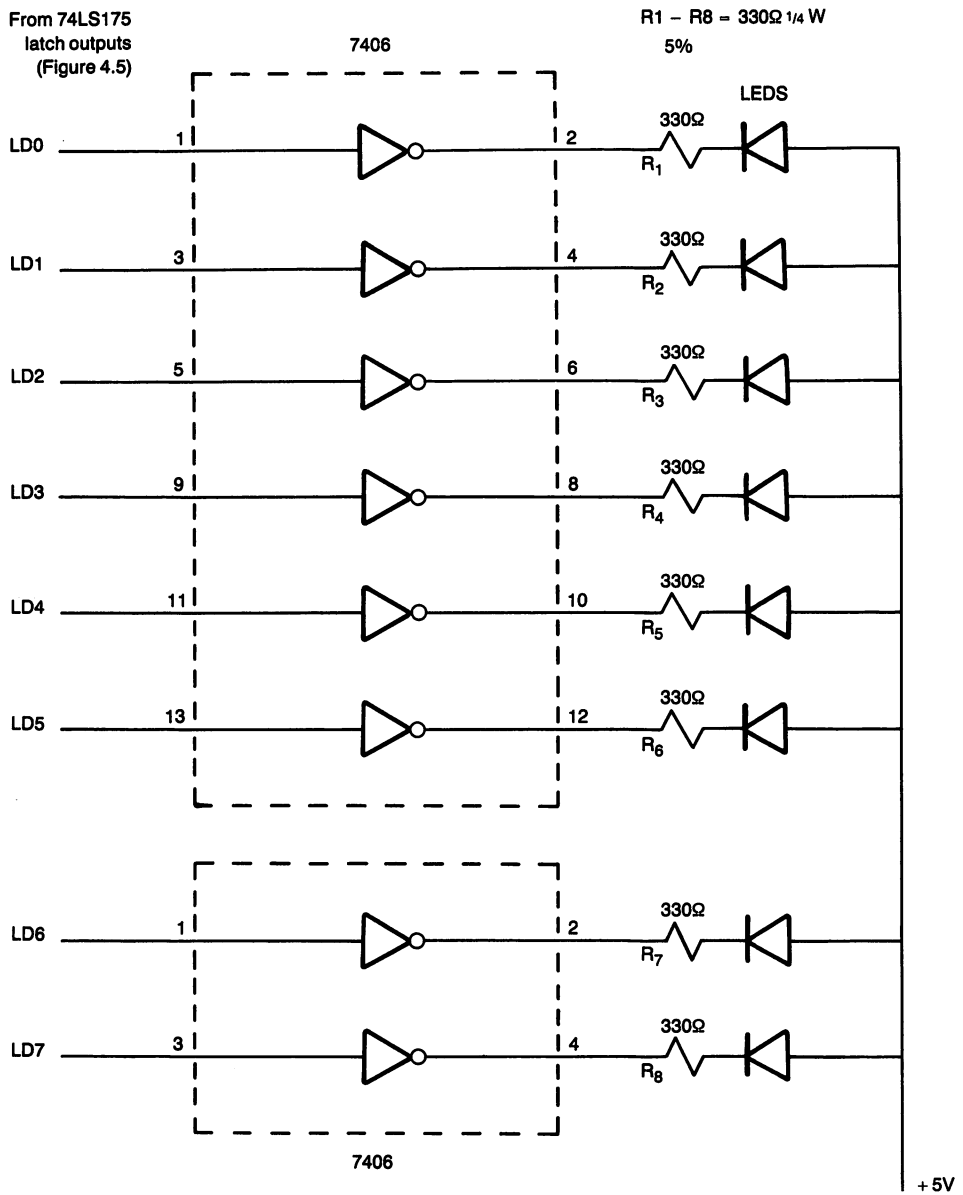



***Figure 4.8**–Schematic diagram of the driving circuits for eight LEDs.*

What we have described and discussed in the preceding sections is the essential idea of output-hardware interfacing. The problem we solved was how to get any amount of information from a specific place to another specific place, with every bit under precisely planned control in order to accomplish a particular task. We placed a logical 1 or a logical 0 on a digital input line, and the result was an action taking place in the output device. In the case just discussed, the action was simply turning on and off an LED. Although this action is simple, it accurately reflects the basic principles of output interfacing.

The inputs to the 7406 shown in Figure 4.8 will come from the outputs of the 74LS374 latch (shown in Figure 4.5) that we discussed earlier. Latched outputs, once set, will remain at a logical 1 or a logical 0 until we write another byte to the device using the POKE instruction. In this way we can turn on and off the LEDs under computer control. A BASIC program can be generated that will write data to the output device, causing the hardware to respond in the manner described. We discussed this type of software in Chapter 2.

The overall result will be to turn on and off different LEDs under our control via the program written. Simple as it appears, this experiment is an excellent place to begin addressing the problem of interfacing different types of external hardware, because it embodies typical elements involved in all present-day computer output transfers of information.

## 4.7 HARDWARE FOR INPUTTING DATA TO THE VIC-20

In the preceding discussion, we examined the hardware for outputting data from the VIC-20 computer. Now, we will turn our attention to inputting data to the VIC-20 from an outside source. That is, some external device will output data; this data will be input to the computer, which can make decisions based on the input data, store the data, or alter the data in any manner we desire. We discussed these operations in Chapter 3. Let us now discuss how to design the hardware that will allow the input of external data to the VIC-20.

The same general comments we made about the output hardware at the beginning of this chapter apply here. This hardware is presented mainly for its educational value, to illustrate the basic concepts

involved in computer interfacing. The hardware shown is very simple, and it does work. Like the output hardware discussed earlier, it has been designed to use standard "off-the-shelf" integrated circuits, so you can actually construct and experiment with it. In fact, experimentation of this type is highly recommended prior to designing any interface circuits.

To input data to a BASIC program, we will use the PEEK instruction described in Chapter 3. The PEEK instruction will use an address to input from. This address will be logically decoded to enable the $\overline{I/O2}$ select line, which we discussed in detail in the output section. That is, whenever the software is performing a PEEK at the correct address for the I/O circuit, the $\overline{I/O2}$ select line will be an active logical 0.

The PEEK instruction will be performing a read function and the VR/$\overline{W}$ line on the I/O expansion connector will therefore be a logical 1. Recall that the VR/$\overline{W}$ line was a logical 0 during the POKE instruction. This line will be a logical 1 during the PEEK. It will therefore be logically inverted with a 74LS04, as shown in Figure 4.9. The VR/$\overline{W}$ and the $\overline{I/O2}$ select line are logically combined to generate the $\overline{\text{BOARD ENABLE}}$ line. This is shown in Figure 4.9.

During the time the VIC-20 is reading the data from the selected I/O circuit, data from that circuit is electrically enabled onto the VIC-20 data lines D0– D7, as shown in Figure 4.10.

Figure 4.11 is a complete schematic diagram of a circuit that will

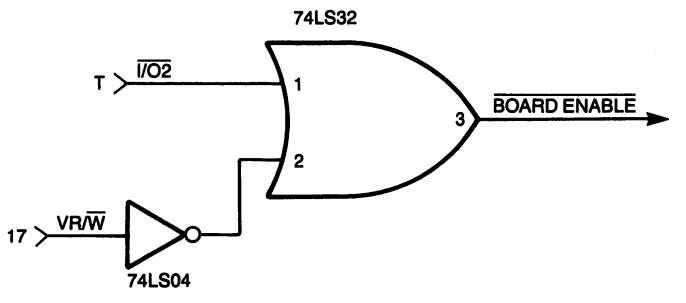



***Figure 4.9**–Schematic diagram of the hardware necessary to qualify the VR/$\overline{W}$ line during an input read operation from the I/O expansion connector.*

input data to the VIC-20 from an external source. This circuit will employ the circuits of Figures 4.9 and 4.10. As you can see from this schematic, there is very little hardware required. Further, the hardware is simple in its operation. Let us discuss exactly how the hardware shown in Figure 4.11 operates.

At the center of Figure 4.11 is the 74LS240 tri-state buffer. This buffer takes input on pins 2, 4, 6, 8, 11, 13, 15 and 17. These inputs are labeled XCD0–XCD7. (The XCD stands for External Card Data,

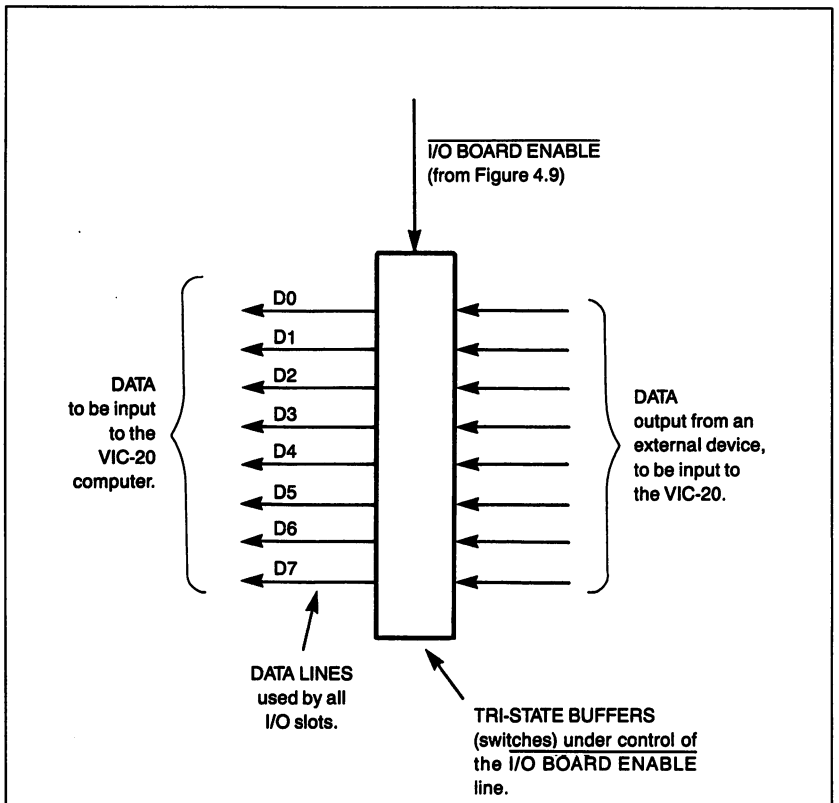



***Figure 4.10**–Block diagram indicating that external data is input to a tri-state buffer. The buffer outputs are connected to the data lines of the VIC-20 computer. The BOARD ENABLE line shown in Figure 4.9 will enable the buffer outputs onto the VIC-20 data bus at the correct time.*

which is the information present on the data lines to be input to the VIC-20.) These inputs are held at either a logical 1 or a logical 0 level. The input pins can hold any digital information desired.

We are not now interested in what data might be input, but for the purpose of discussion we will assume that these input lines actually represent some digital information to be sent to the VIC-20 computer.

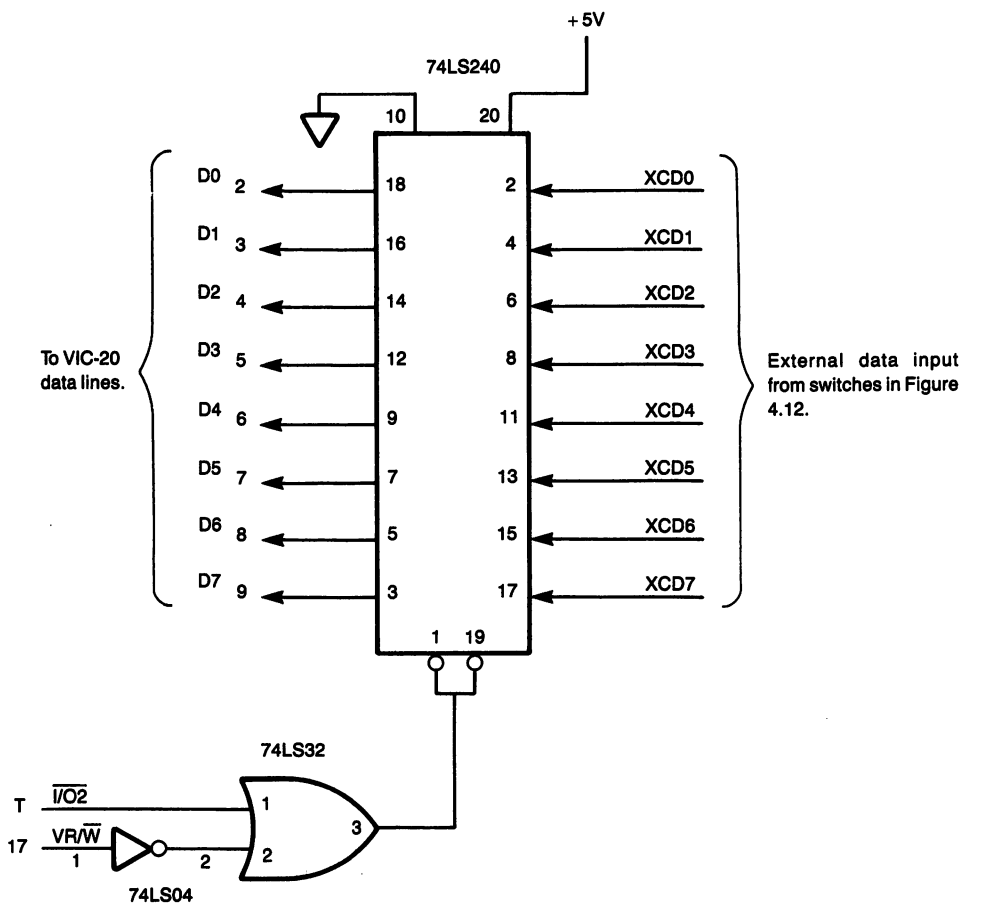



***Figure 4.11**–Complete schematic of the hardware necessary to input an external data byte to the VIC-20 computer from the expansion connector.*

The objective of this discussion is to show *how* that information can be input to the computer. You must be able to master this hardware before proceeding on to the next step in the interfacing process.

In our circuit the lines XCD0–XCD7 will be connected to a DIP switch. DIP stands for *Dual In-line Package*. A schematic of this type of switch is shown in Figure 4.12. We see in this diagram that one side

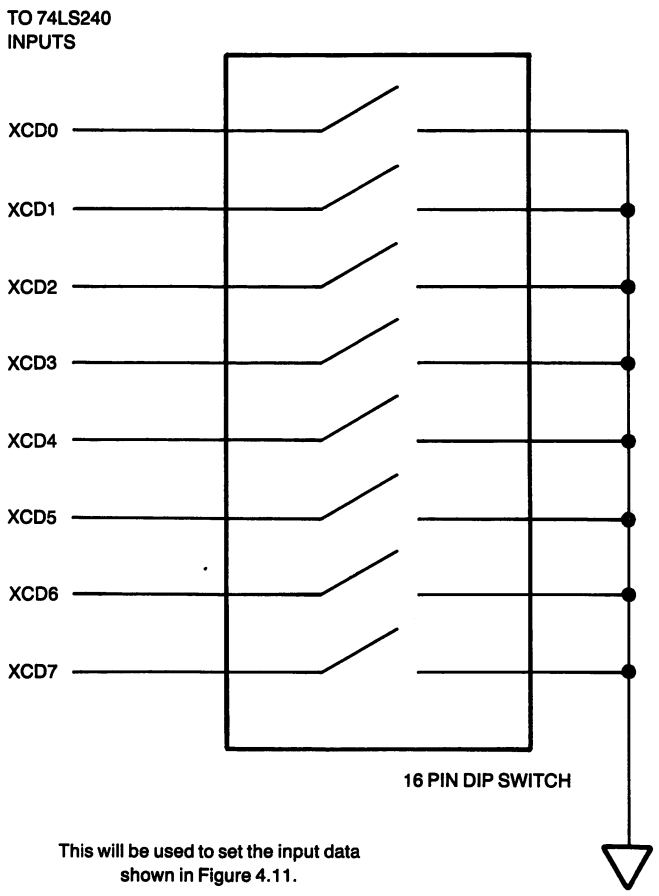



***Figure 4.12**–Schematic diagram of eight switches used to set the logical level of the eight external input lines at the I/O slot.*

of the switch is connected to ground. The other side of the switch is connected to the XCD input lines of the 74LS240 tri-state buffer shown in Figure 4.11. When the switch is closed, the input to the 74LS240 is a logical 0. When it is open, the input to the 74LS240 is a logical 1. We are making use of the fact that an open, or floating, TTL (Transistor-Transistor Logic) input is a logical 1. (TTL is a family of digital electronic circuits. The VIC-20 uses TTL for some of its internal circuits.) A 74LS240 will logically invert the information input prior to outputting data to the VIC-20 data lines.

By using the switch, we can force any of the eight inputs to a logical 1 or a logical 0. This data will be sent to the VIC-20. Note that this is exactly how the CMS I/O system we described in Chapter 3 operated.

The outputs of the 74LS240 shown in Figure 4.11 are connected to the VIC-20 computer data bus lines, D0–D7. The outputs of the 74LS240 must not be enabled or turned on unless the computer is electrically requesting the data from the input circuit. At all other times, the data outputs from the buffers are set into a tri-state, or off, mode. This mode will electrically remove the 74LS240 outputs from the VIC-20 data bus lines. The only time the outputs of the latch will be placed on the data bus is when the input pins 1 and 19 are set to a logical 0. These two pins are connected to the output of the OR gate, and thus will be controlled by the VR/$\overline{W}$ and $\overline{I/O2}$ lines.

## 4.8 ENABLING THE TRI-STATE BUFFER

The output of the 74LS240 buffer will be electrically placed on the data bus only when pins 1 and 19 (the enable pins) are a logical 0. When this occurs, it is because the VIC-20 is electrically requesting data from the selected I/O addresses. This will occur during a PEEK instruction. Let us examine how the enable pins are set to a logical 0 under control of the computer.

When the computer sends out an address that is equal to the address for the I/O circuit, the $\overline{I/O2}$ line will become active, as we saw in our discussion of the mechanics of output. This line is input to pin 1 of the 74LS32 of Figure 4.11. Remember that this line is active during a write or POKE operation also. Therefore, we must use the VR/$\overline{W}$ line as a qualifier. When the VR/$\overline{W}$ line is a logical 1, the VIC-20

is electrically expecting some external device to send data to it. In Figure 4.11, the VR/$\overline{W}$ line is connected to pin 2 of the 74LS32. With both pins 1 and 2 of the 74LS32 a logical 0, pin 3, the output, is also a logical 0. This pin is connected to the enable input pins 1 and 19 of the 74LS240.

In review, the following events will occur during a read or PEEK operation from an I/O circuit:

1. The VIC-20 will output the correct address that will enable the proper $\overline{I/O2}$ line.
2. The VR/$\overline{W}$ line will go to a logical 1. This action will electrically inform the system hardware that the computer is expecting an external device to place data on the system data bus. When the $\overline{I/O2}$ line goes to a logical 0, the enable input pins 1 and 19 of the 74LS240 are set to a logical 0. At this time, the data at the inputs of the buffer are placed on the VIC-20 computer's data bus.
3. During the time the external data is enabled onto the system data bus, the VIC-20 will automatically read the data. In Chapter 3 we discussed how the software will use the data that was read.
4. After a fixed period of time (approximately 1 millionth of a second), the $\overline{I/O2}$ select line will go to a logical 1 state. This event is performed automatically by the circuits in the VIC-20; the user need not be concerned about forcing this event to occur. When it does occur, the enable input pins 1 and 19 of the 74LS240 will be set to a logical 1. This action will tri-state, or turn off, the 74LS240 buffer outputs, which will electrically remove them from the VIC-20 data bus lines.

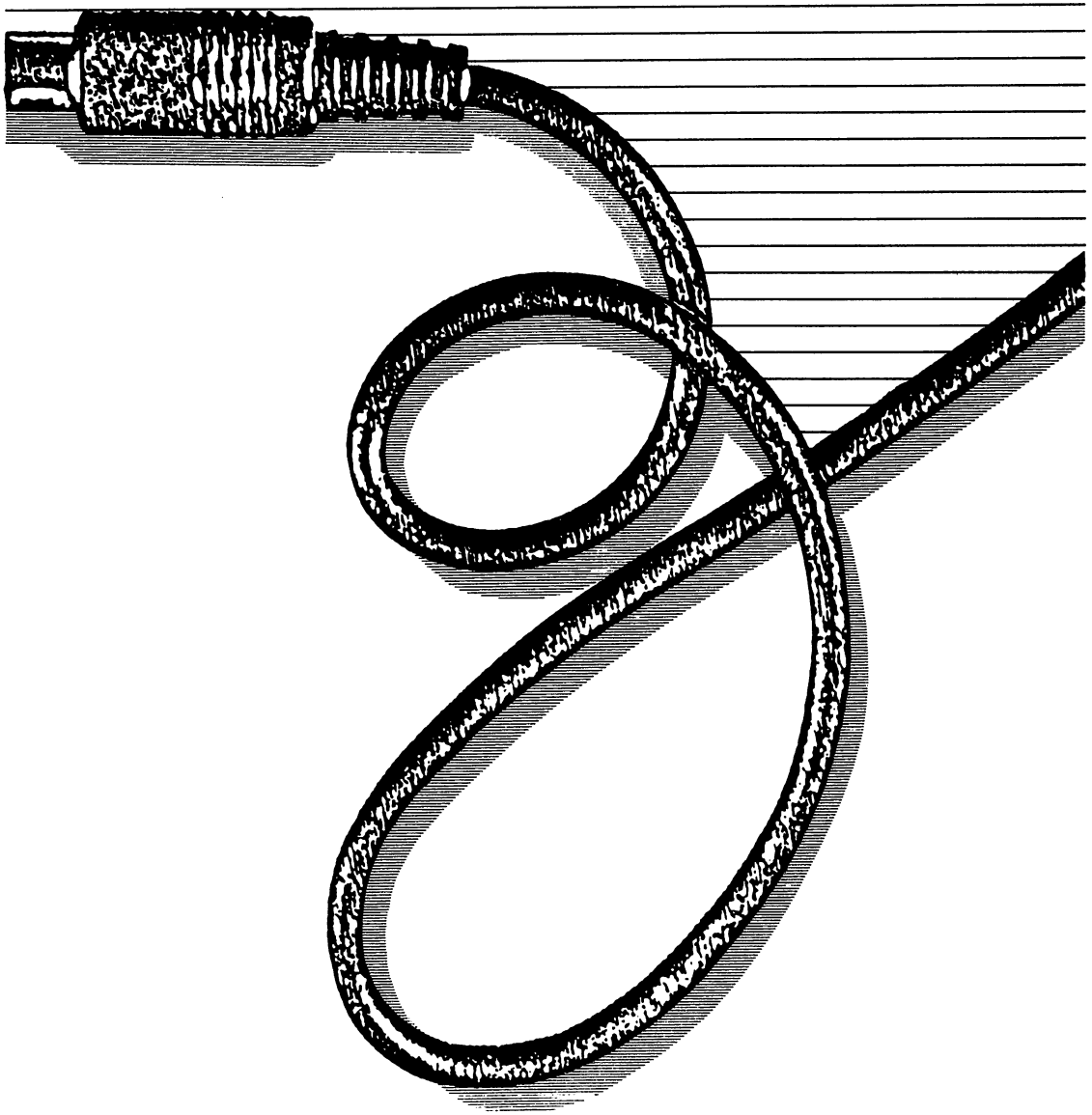
## 4.9 SUMMARY OF INPUT AND OUTPUT

In this chapter we have presented the essential details of inputting and outputting data with the VIC-20 Personal Computer. The schematic diagrams for the actual hardware used were shown and discussed, and you can build this circuit if you want to.

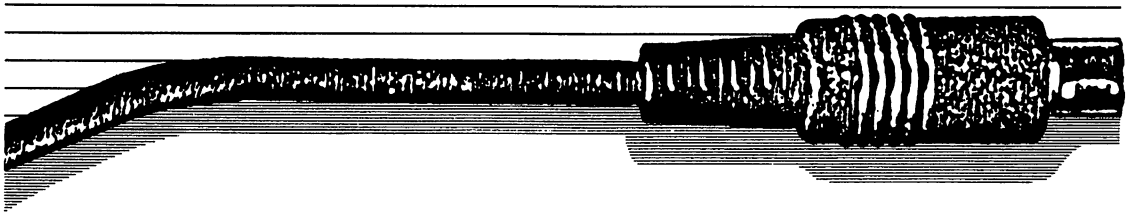
If you understand the information presented in this chapter, you have taken an important step in the interfacing process. The next step will be to connect the input and output lines to different types of external hardware. This will allow the computer to control other hardware besides the LEDs shown here.

In the next chapter we make use of the hardware and software concepts we have explored to design a home security system. In Chapter 6 we will discuss how to use the digital outputs to control a voice for the VIC-20. However, before you attempt to accomplish these tasks, you must have a good understanding of the information presented in this chapter. It is recommended that the beginner in computer interfacing take the time to construct the circuits presented here. All of the hardware components are readily available from many suppliers (see Appendix D), and there are circuit boards available for wiring up circuits that will plug directly into the VIC-20 computer's I/O expansion slots.

# AN APPLICATION OF COMPUTER INTERFACING: A HOME SECURITY SYSTEM



# 5



In this chapter we will present the complete hardware and software necessary for a possible home security system. The controlling software for this application will again be written in BASIC, and we will use information that was given in Chapters 1 – 4. This chapter will help to bring together all of the important points covered in the first four chapters of this text.

The system to be discussed does work, but like the simple circuits presented earlier, its primary purpose is instruction. The intention is to show you one way such a system can be designed. Once the problem has been solved in some manner, adapting the solution to a particular case will be much easier. It is hoped that, after you have been shown one way the VIC-20 can be used to monitor and secure a home, applying this information to your own home will be straightforward, exciting, and fun. You can use this chapter as a starting point for designing other external systems to be controlled by the VIC-20. By the conclusion of the chapter, you will see that this is not too difficult a task, and is a rather enjoyable challenge.

## 5.1 DEFINITION OF THE PROBLEM

Let us start this overall design with a definition of the problem. The first step in any design problem is to define exactly what the system is to do. If the problem is clearly defined at the outset, then we have a direct path to follow. If we do not know exactly what we want from the beginning, it will be extremely difficult to decide as we go along.

A first, general definition is this: *The system to be designed and realized will monitor the status of all doors and windows in a home.* Already we see a problem: there may be some windows that cannot be opened. These windows will not be monitored. Also, there are many doors in a home that do not directly lead to the outside. So let us refine our definition thus: *The system will monitor the status of any door that leads to the outside of the home and any window that can be opened.* Unfortunately, this system definition is still too vague. What exactly is meant by "monitoring the status" of something? In this case, it means indicating whether the door or window is open or closed.

The security system will determine whether a window or door is open or closed. If the system determines that a door or window is open, what action will be taken? That is the next part of the problem we must define. *If a door or window is open, the system will indicate visually on the VIC-20 screen which door or window it is.*

Here, now, is the complete definition of the system.

1. The system will monitor every door in the house that opens directly to the outside.
2. The system will monitor every window in the house that can be opened.
3. If the system detects an open window or door, it will visually indicate this on the CRT screen of the VIC-20.

There are certainly more functions you could design into the system. It depends on how complex you wish the system to become. Some systems have been designed to incorporate a modem and a voice synthesizer to call the police if one of the windows or door is tampered with. This example is given only as an illustration of what can be done. The definition of the system that we have given will enable us to highlight the important details of interfacing and computer control with the VIC-20, without making the task needlessly complex.

## 5.2 DRAWING THE HOUSE WITH THE COMPUTER

Part of the definition of the problem is to indicate on the screen of the VIC-20 PC if any of the doors or windows being monitored is open. This can be done in any number of ways. The technique we will use starts with an outline of the house, similar to an architect's blueprint. This outline will be displayed on the screen of the computer. Each window or door being monitored will be shown. Figure 5.1 shows exactly what we will draw on the screen of the CRT.

Shown in Figure 5.1 is the location of each window and door that is being monitored. Each door and window is given a label. When the system detects that a door or window is open, the label associated

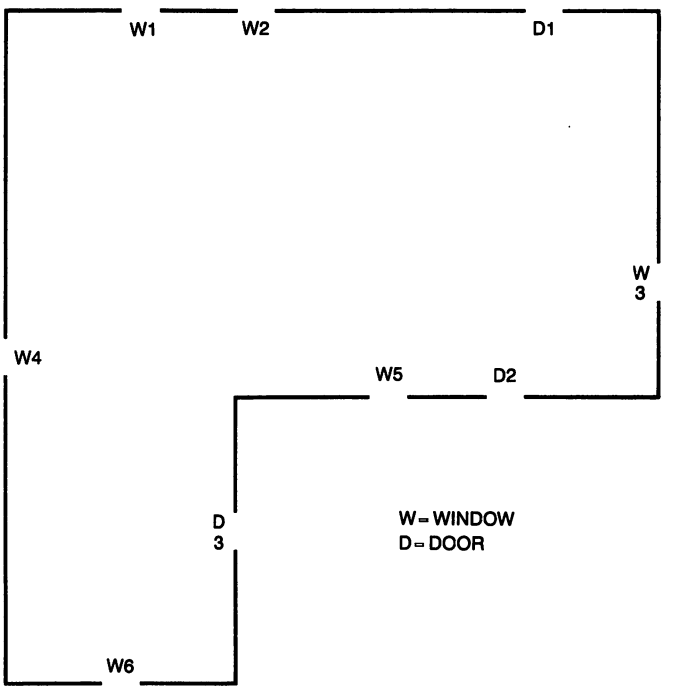



***Figure 5.1**–Outline of the house that will be drawn by the VIC-20 computer.*

with that door or window will be highlighted in a special color on the screen. This will give a quick visual indication of the exact status of any door or window in the house.

Another feature of the program we will write is that the user will have the option of ignoring any open window or door. For example, suppose it is very hot, and you are purposely leaving a window in the bedroom open. The program will ask which windows or doors you wish to ignore. The display on the screen will show that the door or window is open but not generating an alarm. This will be accomplished by highlighting the door or window differently than if it was causing an alarm by being open.

Figure 5.2 shows the first section of the complete BASIC program we will write. This part of the program draws the outline of the house shown in Figure 5.1 on the TV or monitor screen. To do so, it uses some of the special graphics characters available on the VIC-20 computer. Consult your *VIC-20 Programmer's Reference Guide* (Copyright © 1982, Commodore Business Machines, Inc.) for a complete discussion of the use of the VIC-20's graphics characters. You may also want to skip ahead to Figure 5.18 to see a grid of the memory locations used in this program.

```
170  REM DRAW THE SCREEN NOW
180  PRINT CHR$(147)
190  REM SET COLOR OF EACH SCREEN CELL BLACK
200  FOR I = 38400 TO 38884 + 22
210    POKE I,0
220  NEXT I
230  REM TOP LINE OF SCREEN
240  FOR I = 7681 TO 7700
250    POKE I,99
260  NEXT I
270  REM CORNERS
280  POKE 7680,79
290  POKE 7701,80
300  REM SIDES OF HOUSE
```

***Figure 5.2**–BASIC program that will draw the outline of the house shown in Figure 5.1. Special graphics characters of the VIC-20 were used in this program (continues).*

```
310  FOR I = 7702 TO 7944 STEP 22
320    POKE I,101
330    POKE I + 22,103
340  NEXT I
350  REM CORNER + SIDE
360  POKE 7966,101
370  POKE 7987,122
380  REM FIRST BOTTOM LINE
390  FOR I = 7976 TO 7980
400  POKE I,100
410  NEXT I
420  REM SMALLER SIDES
430  FOR I = 7988 TO 8142 STEP 22
440    POKE I,101
450    POKE I + 9,103
460  NEXT I
470  REM CORNERS
480  POKE 8164,76
490  POKE 8173,122
500  REM BOTTOM LINE
510  FOR I = 8165 TO 8172
520  POKE I,100
530  NEXT I
540  REM SCREEN IS NOW DRAWN, NO LABELS IN YET
900  REM WRITE W = WINDOWS, D = DOORS
910  FOR I = 8067 TO 8067 + 7
920    READ V1
930    POKE I,V1
940  NEXT I
950  FOR I = 8089 TO 8089 + 5
960    READ V1
970    POKE I,V1
980  NEXT I
1100 DATA 23,61,23,9,14,4,15,23
1110 DATA 4,61,4,15,15,18
```

*Figure 5.2 (continued).*

## 5.3 PHYSICAL CONNECTIONS TO THE DOORS AND WINDOWS

We have now drawn a layout of the house on the screen of the computer. Next, let us discuss how the physical and electrical connection to the windows and doors is made. We need a device that will transform the motion or position of a window or door into an electrical signal. A common switch will do this.

Various types of switches can be used. The type we will use is installed so that the window or door position will open or close the switch. Figure 5.3 shows a schematic diagram of the type of switch to be used.

We see in Figure 5.3 that when the window or door is closed, its switch will be closed. When the window or door is open, the switch will be open. We now have a device that will transform a physical event, a change in position, into an electrical event, the signal to the computer.

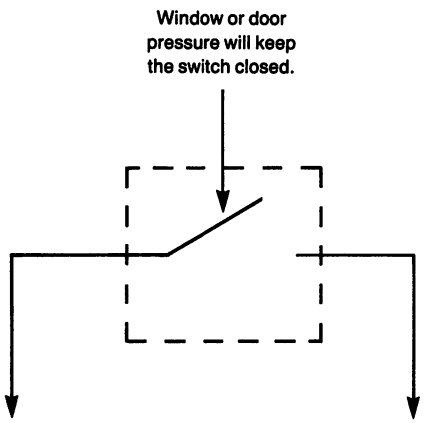



***Figure 5.3**–Diagram showing one type of switch that can be used to monitor doors and windows. The pressure of the door or window will keep the switch closed.*

The electrical quantity the switch uses is resistance. When the switch is closed, corresponding to a closed door or window, the resistance of the switch is approximately zero ohms. When the switch is open, corresponding to an open door or window, the resistance is infinite ohms.

We will use the characteristics of a switch to generate a digital voltage signal that can be input directly to the VIC-20 computer. Figure 5.4 shows how this will be accomplished. One side of the switch will be connected to the ground of the computer. (We will explain exactly how to do this later in the chapter.) The other side of the switch is connected to an input circuit on the VIC-20, exactly like the circuit described in Chapter 4.

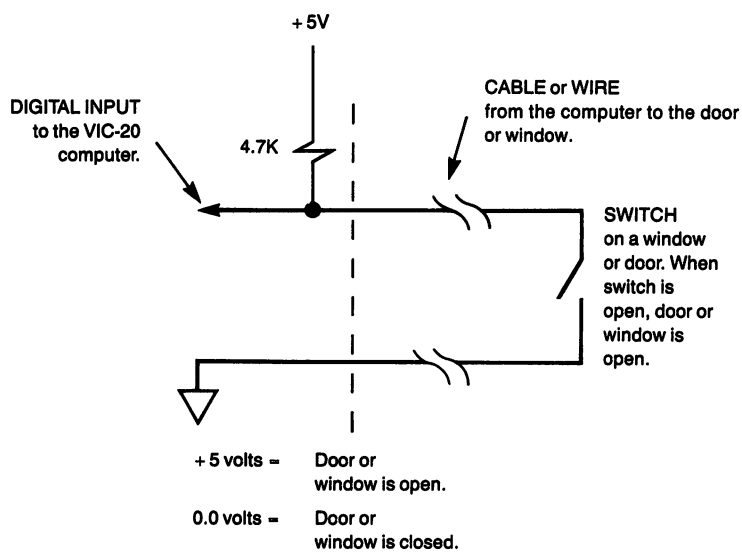



***Figure 5.4**–Schematic diagram showing how the switch will be wired. One side of the switch will be connected to the VIC-20 computer ground, and the other side will be connected to a 4.7K-ohm resistor. It does not matter which side of the switch is connected to ground.*

When the switch in Figure 5.4 is open, there will be no electrical path to ground through the switch. This will allow the input line connected to the 4.7K-ohm resistor to be pulled up to + 5 volts. This voltage level corresponds to a logical 1 voltage level in the VIC-20. When the window or door is closed, the switch will close. This forces the input line connected to the resistor to ground. Now there is an electrical path established through the switch to ground. A voltage of ground potential is equal to a logical 0 in the VIC-20. Another way of saying this is that when the door or window is open, the digital input line to the VIC-20 computer is equal to + 5 volts, which is a logical 1. When the door or window is closed, the digital input line to the computer will equal 0.0 volts, or a logical 0.

Knowing this will enable us to examine the status of each door or window under software control. We can do this using the PEEK instruction and the software discussed in Chapter 3. Later, we will give the complete program for controlling the system.

We have discussed in general terms how the switch will be placed on the door or window to be either opened or closed by its position. Unfortunately, we cannot give an exact, detailed description of how to install these switches in your home, because everyone's windows and doors, and the type of switches used, may be slightly different.

However, there is a positive side to this. It gives you, the user, complete freedom to decide exactly how you want to install the system in your home. It has been the author's experience that the end user will usually improve on any method described in a text. Therefore, this book will give general guidelines to get you started in the correct direction.

If you have a number of doors and windows that you wish to monitor, there will need to be quite a few electrical connections to the VIC-20. The voltages and currents in the circuits are very low, so you can use a light gauge of wire in the connections. What is sold as "speaker wire" has been used very successfully in an application such as this.

Figure 5.5 shows a block diagram of exactly what is occurring when one connects the switches to the doors and windows. We see in this diagram that each switch requires two wires connected to it. This means that for every switch installed, you will have two wires running back to the computer. This could be a bit cumbersome.

Figure 5.6 shows one way to reduce the number of wires that need to be connected to the computer. The idea behind this diagram is to

connect the ground lines of all of the switches together. After the grounds are connected, a single ground wire can be connected back to the computer. It should be noted that it makes no electrical difference which side of the switch is connected to ground.

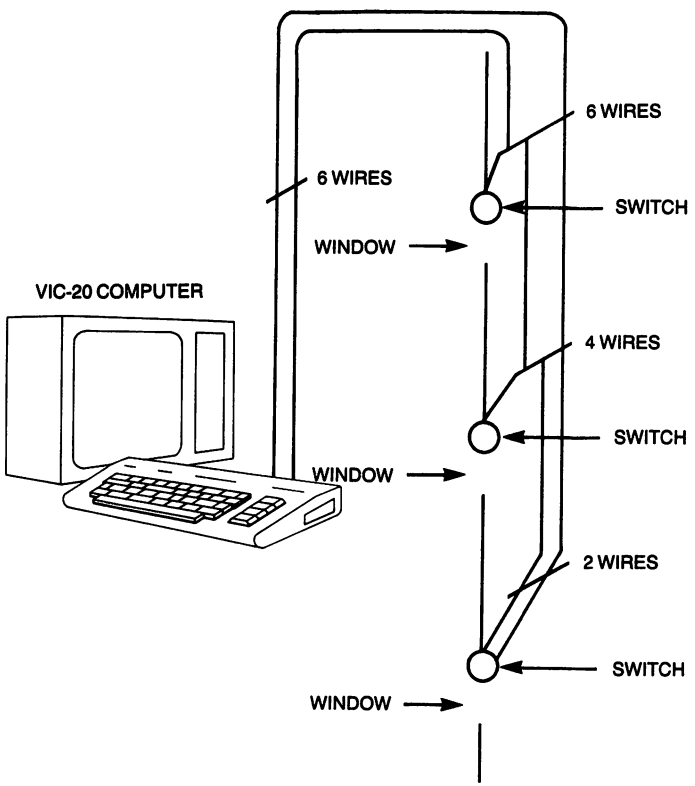



***Figure 5.5**–Pictorial diagram indicating that each switch to be monitored will require two wires sent back to the VIC-20 computer. There are three switches shown here.*

## 5.4 CONNECTING THE HARDWARE TO THE COMPUTER

We now have a bundle of wires from the various door and window switches ready to be input to the computer. The next step would seem to be to electrically connect the wires to the VIC-20 computer. However, before we discuss how to do that, we should examine how to verify that all the wiring to the switches is correct. We want to be sure that all of the switches will open and close, and we want to determine whether or not the wiring from the switches to the computer is complete.

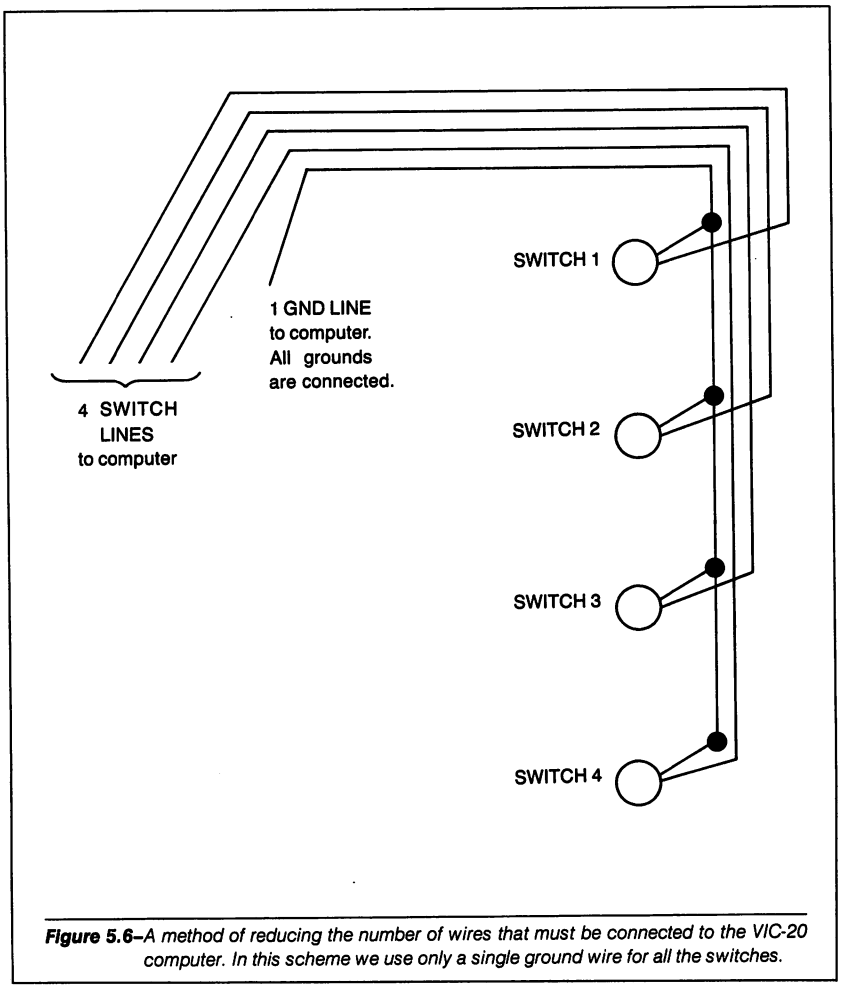



***Figure 5.6**–A method of reducing the number of wires that must be connected to the VIC-20 computer. In this scheme we use only a single ground wire for all the switches.*

This verification procedure involves using either an ohmmeter or a continuity light. An ohmmeter is an instrument that measures the resistance in a circuit. In this application, a closed connection will be approximately zero ohms and an open connection will be (theoretically) infinite ohms. A continuity light is a light bulb that will turn on when the resistance is approximately zero ohms, indicating a closed connection. The light will remain off for an open connection. With either device, the test leads are placed across the two wires that are connected to any of the switches just installed. When the door or window is opened and then closed, the ohmmeter or continuity light will visually inform you if the switch located at the door or window is opening and closing at the correct time. Figure 5.7 shows in pictorial form what we are accomplishing.

If the preceding verification is valid for every switch, we know that everything is operational up to the point where the lines are connected to the computer. We will now discuss how these lines are input to the VIC-20 computer. Figure 5.8 shows the electrical connections and the digital electronics necessary to input the signal lines to the computer.

Let us discuss this schematic diagram in detail. On the right-hand side of Figure 5.8 are the electrical input wires from all of the switches, labels W1 – W6 (the windows) and D1 – D3 (the doors). One side of each switch is connected to a particular input pin of one of the two 74LS244 buffers, labeled IC3 and IC4. Remember that this line is also connected to a 4.7K-ohm resistor, as shown in Figure 5.4. The other side of each switch wire is connected to the ground pins on the VIC-20 computer expansion slot.

Let us now concentrate on exactly how the "W" and "D" input lines are read by the computer. In Chapter 4 we discussed the hardware necessary to allow data to be input to the VIC-20 computer. We have essentially duplicated those circuits given in Chapter 4 twice for this application.

The data will be read into the computer with the PEEK instruction. One main difference between this circuit and the circuit discussed in Chapter 4 is that two lines, labeled A1 and A2, have been added. The lines are input to 74LS00 NAND gates, IC1 of Figure 5.8. These two lines are called address lines. They allow us to have several addresses associated with each I/O circuit in the VIC-20. These lines were not used in Chapter 4, because at that time we only used one address per

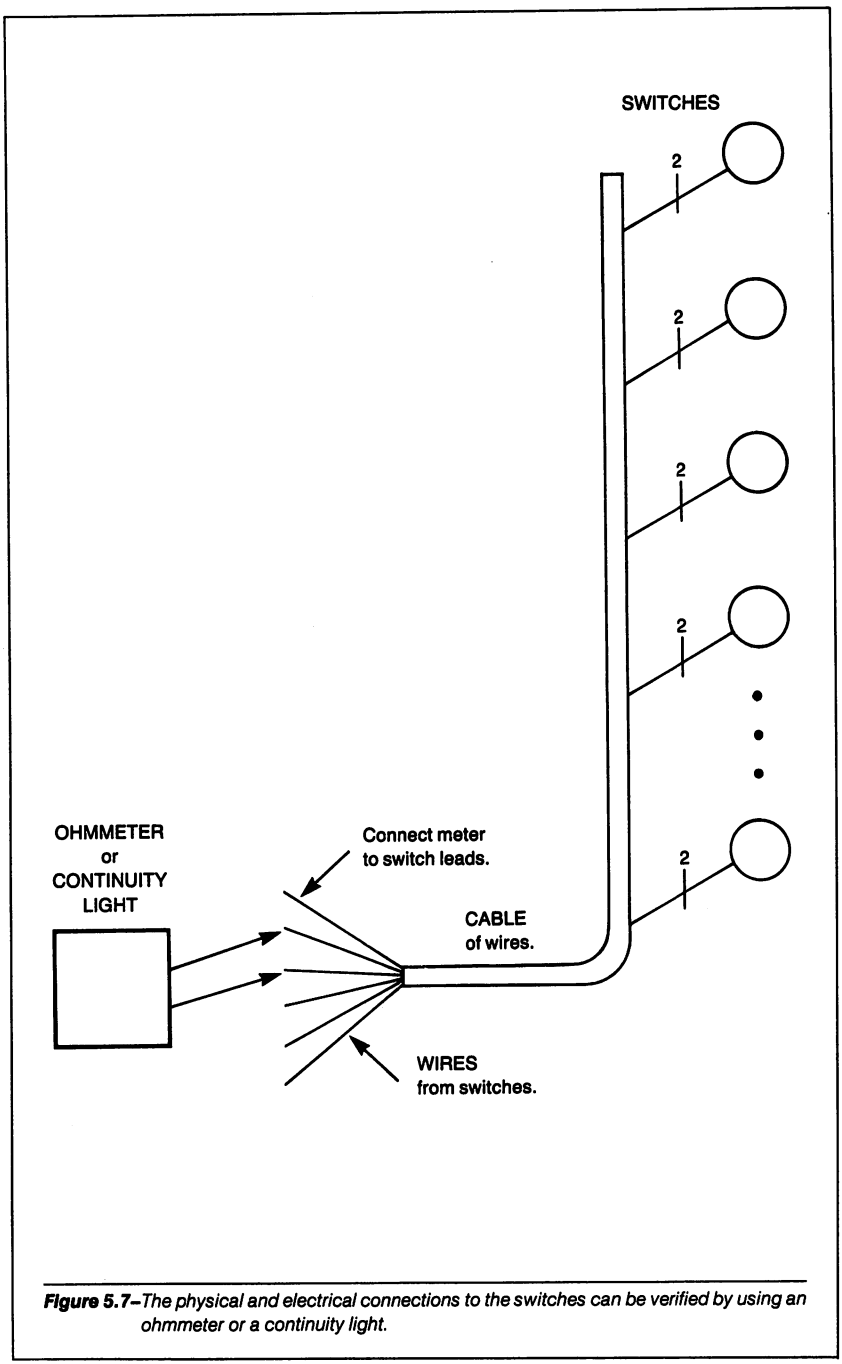



***Figure 5.7**–The physical and electrical connections to the switches can be verified by using an ohmmeter or a continuity light.*

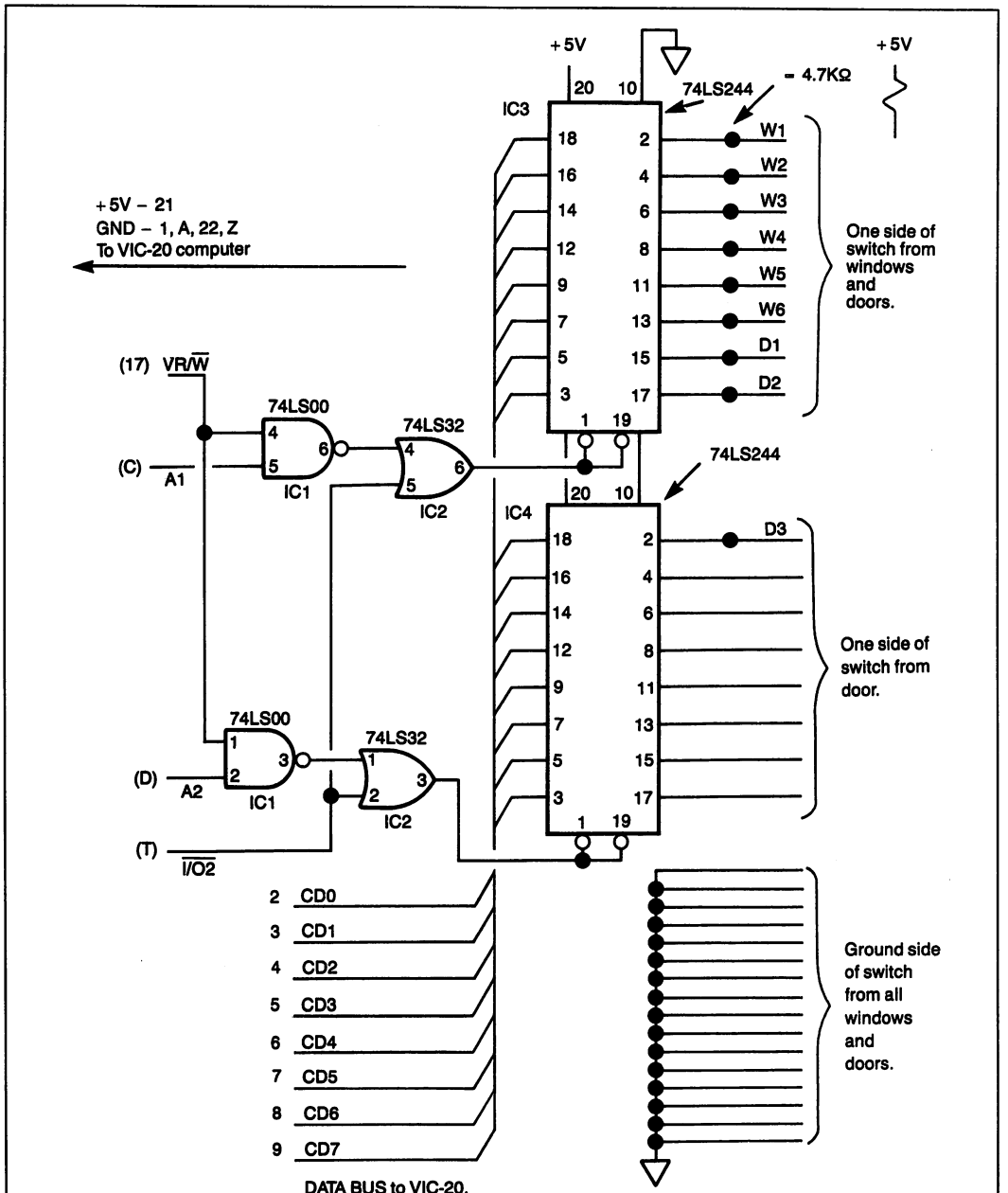



***Figure 5.8**–Complete schematic for the I/O circuit used to connect the VIC-20 computer to the external switches on the doors and windows.*

I/O circuit. Now we have two buffers, and we need two addresses.

The new address for enabling IC3 of Figure 5.8 will be equal to:

```
PEEK address = I/O address + 2
```

Adding 2 sets address line A1 to a logical 1. To enable IC4 of Figure 5.8, the PEEK address will be equal to:

```
PEEK address = I/O address + 4
```

Adding 4 sets address line A2 to a logical 1. For example, if we want to read the logical level of the input lines connected to IC3 of Figure 5.8, we could do it as follows. (We will assume the circuit has an I/O address of 38912.) In BASIC the instruction would be:

```
B1 = PEEK (38912 + 2)
```

or

```
B1 = PEEK (38914)
```

To read the logical levels of the input lines connected to IC4, we would use the BASIC statement:

```
B2 = PEEK (38912 + 4)
```

or

```
B2 = PEEK (38916)
```

At the conclusion of these two instructions, the variables B1 and B2 would be equal to the input weights associated with the open and closed position of each door and window. What must be done now is to write the BASIC language software that will examine the position of each door and window individually.

## 5.5 SOFTWARE FOR INTERPRETATION OF THE INPUT LINES

Now that we have an electrical means of entering the logical conditions of the windows and doors into the program, what do we do next? We must pause here and realize exactly where we stand. At this point in the design of our security system, we are able to input into a BASIC variable the logical condition of each switch in the home. That is, we are able to determine with software whether a window or door is open or closed.

From this point on, what is done with that information depends on the imagination of the user. In our case, we want to inform the user whether any particular window or door is open or closed. The point to be stressed at this time is that you are free to do with the data what you please. This fact leaves the territory wide open for applying any of the software graphics and clever tricks you may know how to do.

Figure 5.9 shows a block diagram flowchart of exactly what the software to be written next will do. In this flowchart the tasks shown are large ones, corresponding to the subroutines we will write. In the first block, the outline of the house is drawn on the screen. In the second, the PEEK instruction is used to read the status of the digital signals connected to the windows and doors into the VIC-20 computer.

In the following block (step 3), an array called "DATA ARRAY" will be filled. Each element of this array will have a 1 or 0 in it, based on

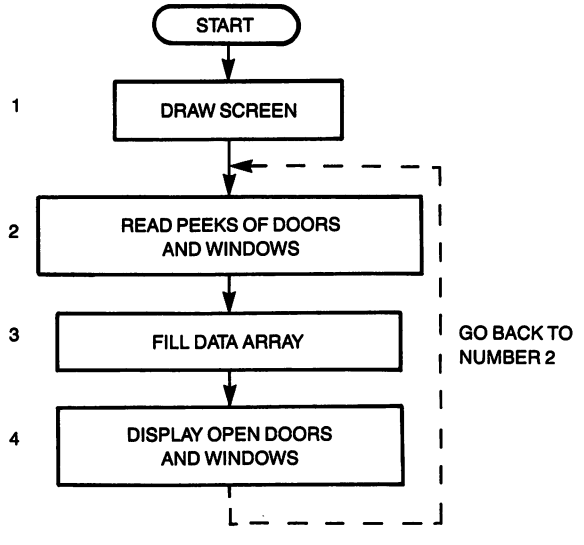



***Figure 5.9**–Task flowchart for the major jobs that must be accomplished with the control software.*

the logical value of the corresponding window or door that is being monitored. In our data array there will be 16 elements, D(1) to D(16), of which only 9 are used at this time. The hardware allows 16, in the event that you wish to expand your system. Each element will correspond to the logical input value of the signal line shown in Figure 5.8. For example, D(1) equals the logical value of the W1 input from the external window monitor, D(2) equals the logical value of the W2 input, D(3) equals the logical value of the W3 input, and so on.

Finally, the software will display the status of each window and door on the screen.

These are the major software tasks that need to be accomplished in order to complete the design of our system. In this section we will discuss the software to implement the second and third blocks of the flowchart in Figure 5.9. We have already discussed how to draw the house on the screen of the computer. The two blocks to be discussed will read the data at the I/O circuit and then fill the data array with the proper 1s and 0s.

We have already seen in Section 4.7 how to read data into the computer from an external I/O circuit. However, it might be a good idea to summarize the important points of that discussion here. The BASIC instruction used to input the data is PEEK. There will be two PEEK instructions necessary to read the entire 16 data lines that were shown in Figure 5.8. (Recall, however, that we are only using 9 of the possible 16 in this application.)

The data read into the computer from the input slot will be stored in an array. We will name the array "A" in order to allow for any future expansion of the number of input lines for the system. Array A will have only two elements at this time, A(1) and A(2).

A(1) will store the summed weight of the data lines from IC3 of Figure 5.8, and A(2) will store the summed weight of the data lines from IC4 of Figure 5.8. (These integrated circuits are 74LS244s, octal buffers with eight output lines each.) Recall that the summed weight can be any number from 0 to 255, inclusive.

You should also remember that the PEEK address is dependent on the address of the I/O circuit. In the program we are writing, we will use the more general name "I/O add" to mean the number that corresponds to the I/O address of a particular circuit in the VIC-20 PC. For the program to actually work, of course, you would need to use a *real* address. The two PEEK instructions used are:

A(1) = **PEEK** (I/O add + 2)

and

A(2) = **PEEK**(I/O add + 4)

These two instructions will store the summed weight of the 16 data input lines in the array A.

Data has now been entered into the BASIC program. The next step will be to fill the data array D with the proper 1s and 0s. The list of the corresponding D array elements and the window or door each represents is shown in Figure 5.10.

The section of software for filling the data array with the correct values is shown in Figure 5.11. We will first show the entire routine and then break each instruction down and provide further explanation where it is required.

| Data Array Value | | Window or Door |
|---|---|---|
| D(1) | = | W1 |
| D(2) | = | W2 |
| D(3) | = | W3 |
| D(4) | = | W4 |
| D(5) | = | W5 |
| D(6) | = | W6 |
| D(7) | = | D1 |
| D(8) | = | D2 |
| D(9) | = | D3 |
| D(10) | = | NOT USED |
| D(11) | = | NOT USED |
| D(12) | = | NOT USED |
| D(13) | = | NOT USED |
| D(14) | = | NOT USED |
| D(15) | = | NOT USED |
| D(16) | = | NOT USED |

***Figure 5.10**–List of the "D" (data) array elements and the corresponding "W" or "D" input line each one represents with software.*

In lines 500 and 510 the variable R2 will be set equal to the number of the A array element set by the value of T. T will determine which array element, 1 or 2, we are using. This subroutine will be called for each element of the A array. In our program this subroutine will be called twice: once with T equal to 1, and again with T equal to 2.

The instruction:

```
520 FOR I = F TO F + 7
```

will be the start of the FOR/NEXT loop. The variable F will be set prior to calling this subroutine, and will determine the starting element of the data array. Each time the subroutine is called, eight elements of the data array are filled. The first call or GOSUB will be with F equal to 1, the next GOSUB will be with F equal to 9.

```
530 IF R2 - R1 < 0 THEN 560
540 R2 = R2 - R1
550 D(I) = 1
```

The value of D(I) was originally set to zero. It will be changed by the weight of any D array element tested, if that weight was used in the summation. This is exactly the same type of operation we described in section 3.5. A program similar to this was shown in Figure 3.9.

```
560 R1 = R1/2
570 NEXT I
580 RETURN
```

```
500 R1=128
510 R2=A(T)
520 FOR I=F TO F+7
530   IF R2-R1<0 THEN 560
540   R2=R2-R1
550   D(I)=1
560   R1=R1/2
570 NEXT I
580 RETURN
```

***Figure 5.11**–This subroutine will fill the data array.*

Line 560 will compute a new value of R1, which will be the weight of the next lower data input line to test. R1 will start at 128, which is the weight of D7. R1 is set to 128 in line 500 of the subroutine.

The way in which the subroutine just described will be called by the main program is shown in Figure 5.12.

At this time the data has been input and the data array D is filled with the corresponding 1s and 0s for the logical inputs of the windows and doors being monitored by the computer.

## 5.6 SIMULATION OF ALL WINDOWS AND DOORS FOR PROGRAM DEVELOPMENT

Now we are ready to start writing the software to display the results on the screen. This type of software development is an empirical, trial-and-error process. That is, we make a first pass at what we think the output display should look like, then, based on what we see, we adjust the program to make the output more visually pleasing.

When we are doing this type of program development, it is useful to be able to simulate the condition being monitored—in this case, the

```
 95  REM READ WINDOWS AND DOORS (LINES 100 - 110)
100  A(1) = PEEK(I/O add + 2)
110  A(2) = PEEK(I/O add + 4)
115  REM SET D ARRAY EQUAL TO ZERO (LINES 120 - 140)
120  FOR I = 1 TO 16
130    D(I) = 0
140  NEXT I
150  T = 1
160  F = 1
170  GOSUB 500 REM CALL THE SUBROUTINE
180  T = 2
190  F = 9
200  GOSUB 500 REM CALL THE SUBROUTINE AGAIN
```

***Figure 5.12**–A section of the main program with the subroutine calls.*

opening and closing of any combination of doors and windows. Of course, we could simply go into the room where the door or window is and open or close it. However, the computer is usually located away from the site being monitored. It would be tedious and frustrating to have to get up from the computer each time we wanted to try a different combination of settings.

There are two possible ways to remedy this situation; one involves hardware, and the other involves software. The objective of each technique is to fill the data array, D, with any combination of 1s or 0s that we wish. This will simulate any combination of windows or doors being open or closed.

The hardware technique involves building a switch box that will connect to the I/O circuit in place of the cable from the switches located in the doors and windows. This method is shown in Figure 5.13. Using this technique, we can simply run the existing program and flip a switch on the box to change the status of any door or window being monitored.

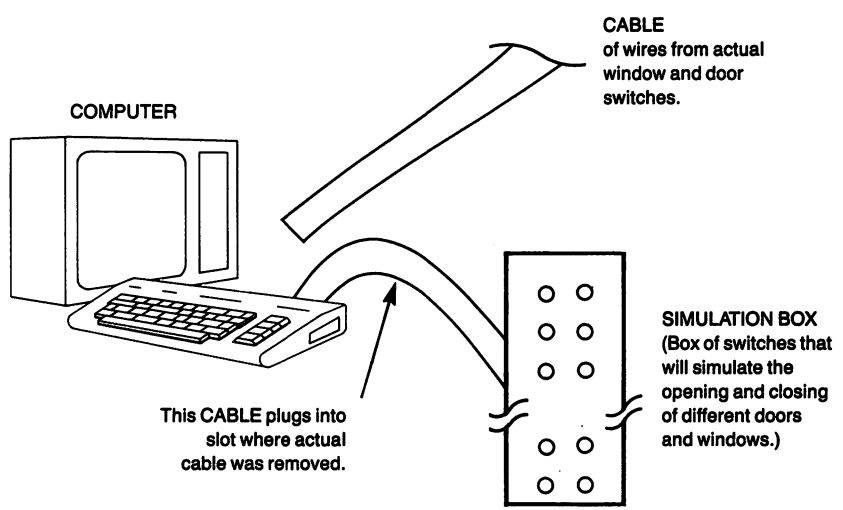



***Figure 5.13**–The cable from the external window and door switches can be disconnected from the VIC-20 computer and a simulation "switch" box can be connected in its place. This type of hardware box can be used for software development, and for system debugging.*

The software technique for simulation can be as exotic as you wish. However, it should be kept in mind that this is a short-term effort. The program will not be used much, if at all, once the system is developed. Let this fact guide you in the amount of time spent on its creation.

The program we will show asks the user what the value of each element of the D array is to be. The user will enter the value 1 or 0 for each element. In some cases the user will want to change only one of the array values. This program has the ability to do this also. After the D array values are set, the program will jump directly to the section of the main program that will output the display. This program is appended to the main program during the software development stage. When the development is complete, the simulation program may be deleted. This program is shown in Figure 5.14.

```
 10 DIM D(16)
 20 PRINT "DO YOU WANT TO CHANGE ALL OF THE VALUES"
 30 INPUT G$
 40 IF G$ = "Y" THEN 100
 50 PRINT "ENTER THE DATA ARRAY NUMBER TO TOGGLE"
 60 INPUT Y
 70 IF D(Y) = 1 THEN 85
 75 D(Y) = 1
 80 GOTO 200
 85 D(Y) = 0
 90 GOTO 200
100 FOR I = 1 TO 16
110   PRINT "DO YOU WANT D(";I;") = 1"
120   INPUT G$
130   IF G$ = "Y" THEN 160
140   D(I) = 0
150   GOTO 170
160   D(I) = 1
170 NEXT I
200 REM GOTO main program
```

***Figure 5.14**–This subroutine will simulate the opening and closing of various doors and windows. When software development is completed, you can delete these lines.*

At line 200, the final GOTO, you would jump to the location in the main program where the system is outputting the screen information based on the values in the data array.

If you choose to perform the simulation using hardware, you can follow the schematic shown in Figure 5.15. We see in this diagram

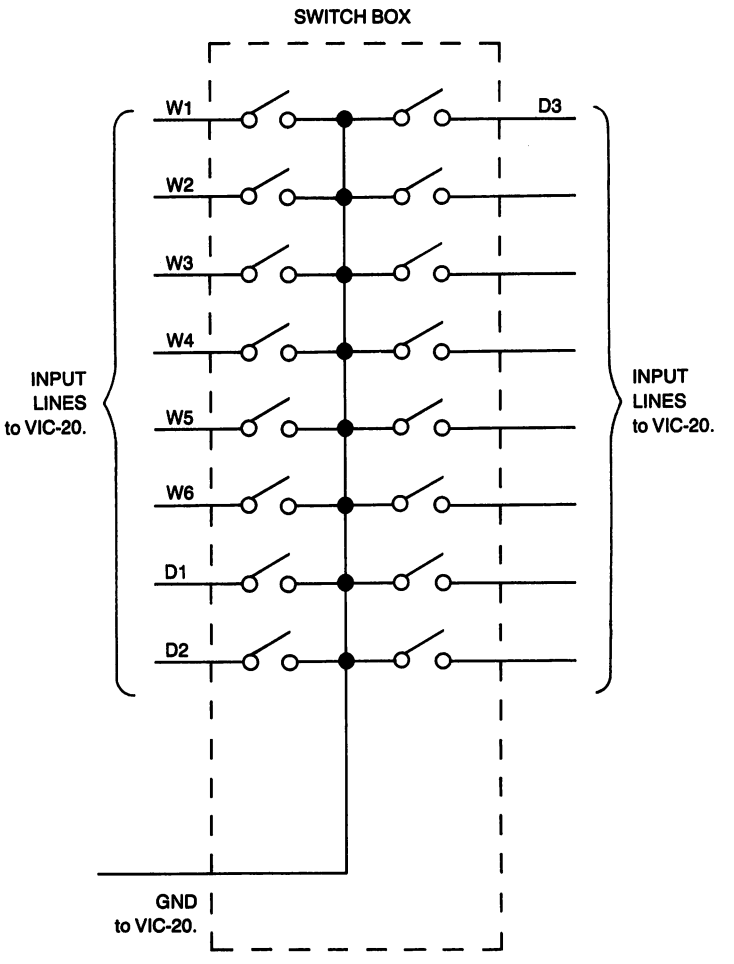



***Figure 5.15**–Schematic diagram of a hardware switch box.*

that each switch corresponds to one of the windows or doors being examined in the system. This type of simulation also has the advantage that it can be used as a debugging tool in the event your system become defective. The simulation box can be installed and used to verify that all of the interface hardware in the I/O slot is operational.

## 5.7 MASKING OFF THE ALARMS WITH SOFTWARE

Sometimes, you will not want the fact that a window or door is open to set off an alarm. For example, when it is very hot, you may wish to leave a window or door open. It is possible to mask off any window or door that you do not wish to cause an alarm. (If you design a more elaborate security system with a sound alarm, or even a connection to the police, the masking capability will be especially important.) In this section we will write the software to do that.

The program will ask the user which window or door number to mask off. The results of these questions will reside in an array labeled M. This array will have as many elements as the data array we discussed earlier. If an element of the mask, or M, array is equal to a logical 0 (that is, if the user types "N"), then the alarm is not masked. If the entry is equal to a logical 1, then the alarm is masked. For example, if we wanted to mask the alarm on door D1, then M(7) = 1. The program is shown in Figure 5.16.

```
 10 DIM M(16)
 15 REM INITIALIZE THE MASK ARRAY TO ZERO (UNMASK ALL ALARMS)
 20 FOR I = 1 TO 16
 30   M(I) = 0
 40 NEXT I
 50 FOR I = 1 TO 16
 60   PRINT "DO YOU WANT TO MASK OFF M(";I;")"
 70   INPUT A$
 80   IF A$ = "N" THEN 100
 90   M(I) = 1
100 NEXT I
```

***Figure 5.16**–This routine will mask off any alarm you choose.*

At the end of this program, all of the masks will be set. Remember that if the entry into the M array is a logical 1, then the mask is set. If the entry is a logical 0, then the mask is cleared. This array will be used when the computer generates the correct display on the VIC-20 computer screen.

## 5.8 THE COMPLETE SYSTEM

So far in this chapter, we have separately presented most elements of the software and hardware for the system. In this section we will bring all of these elements together and show the software for the entire system. The software will be broken into its functional parts. Each part will be nearly the same as a previous section we have presented. The new sections of the program will be explained fully. The program shown in Figure 5.20 comprises the software to complete the entire system. First, let us look at the program so far, shown in Figure 5.17. Statements 10–640 will ask for masked alarms to be input, and then draw the outline of the house on the screen. Finally, in lines 650–750, the system will input the status of the windows and doors. This will be the print section to fill in the display outline for the screen.

```
 10  DIM A(5),D(16),M(16),A$(3)
 20  PRINT CHR$(147)
 30  FOR I = 1 TO 16
 40    D(I) = 0
 50    M(I) = 0
 60  NEXT I
 70  REM THE ABOVE ZEROED OUT ARRAYS
 80  PRINT "DO YOU WANT TO MASK ANY DOOR OR WINDOW?"
 90  INPUT A$
100  IF A$ = "N" THEN 180
110  FOR I = 1 TO 16
120    PRINT "DO YOU WANT TO MASK M(";I;")";
130    INPUT A$
```

***Figure 5.17**-The program so far (continues).*

```
140    IF A$ = "N" THEN 180
150    M(I) = 1
160  NEXT I
170  REM DRAW THE SCREEN NOW
180  PRINT CHR$(147)
190  REM SET COLOR OF EACH SCREEN CELL BLACK
200  FOR I = 38400 TO 38884 + 22
210    POKE I,0
220  NEXT I
230  REM TOP LINE OF SCREEN
240  FOR I = 7681 TO 7700
250    POKE I,99
260  NEXT I
270  REM CORNERS
280  POKE 7680,79
290  POKE 7701,80
300  REM SIDES OF HOUSE
310  FOR I = 7702 TO 7944 STEP 22
320    POKE I,101
330    POKE I + 22,103
340  NEXT I
350  REM CORNER + SIDE
360  POKE 7966,101
370  POKE 7987,122
380  REM FIRST BOTTOM LINE
390  FOR I = 7976 TO 7980
400    POKE I,100
410  NEXT I
420  REM SMALLER SIDES
430  FOR I = 7988 TO 8142 STEP 22
440    POKE I,101
450    POKE I + 9,103
460  NEXT I
470  REM CORNERS
480  POKE 8164,76
```

*Figure 5.17 (continued).*

```
490  POKE 8173,122
500  REM BOTTOM LINE
510  FOR I = 8165 TO 8172
520    POKE I,100
530  NEXT I
540  REM SCREEN IS NOW DRAWN, NO LABELS IN YET
550  REM NOW TO GET THE PEEK DATA
560  A(1) = PEEK (38912 + 2)
570  A(2) = PEEK (38912 + 4)
580  T = 1
590  F = 1
600  GOSUB 670
610  T = 2
620  F = 9
630  GOSUB 670
640  GOTO 770
650  REM SUBROUTINE TO FILL DATA ARRAY
660  REM
670  R1 = 128
680  R2 = A(T)
690  FOR I = F TO F + 7
700    IF R2 - R1 < 0 THEN 730
710    R2 = R2 - R1
720    D(I) = 1
730    R1 = R1/2
740  NEXT I
750  RETURN
```

*Figure 5.17 (continued).*

The following will be a new section of code. This section will compare the logical input data from the windows and doors with the mask data and, based on this comparison, will write the appropriate information to the screen. There are three possible conditions to write to

the display:

1. ALARM. The display will show the window or door label in red.
2. ALARM but MASKED. The display will show the window or door label in yellow.
3. NO ALARM. The display will show the window or door number in green.

To set up this new section of code, several details must be taken care of. First, we have not yet labeled the various doors and windows on our screen display of the outline of the house. We must now assign the letters and numbers that will define the labels for each door or window. In our system we have used two letters for this purpose: windows are labeled W1, W2, etc., and doors are labeled D1, D2, and D3. These labels will be incorporated into data statements.

It should be pointed out that the method we will describe is only one of many ways to accomplish this labeling. The data statements for each label will be formatted like this:

```
DATA  23   7684   49   7685
      ↓    ↓      ↓    ↓
      W    mem    1    mem
           add          add
```

where each label has four data parts, two for each character in the label. The first part is the decimal equivalent of the first character; this will be a W or D. Following this will be the memory address of the video location of the letter. Figure 5.18 shows the memory locations and layout of the VIC-20 video memory. The second character also has two numbers associated with it. These four values will be used to POKE the data into the screen memory. The above data statement thus stores in memory the label W1, and its locations, the fourth and fifth columns of the top row of the screen.

There are nine labels: W1 – W6 and D1, D2, and D3. Therefore, we will use nine data statements. Each statement will contain the information necessary for one label. You do not have to use nine data statements, but it makes the organization of the data much simpler in this case.

After the label is written into the screen location, we must set the correct color. The colors used for this application are green, red, and

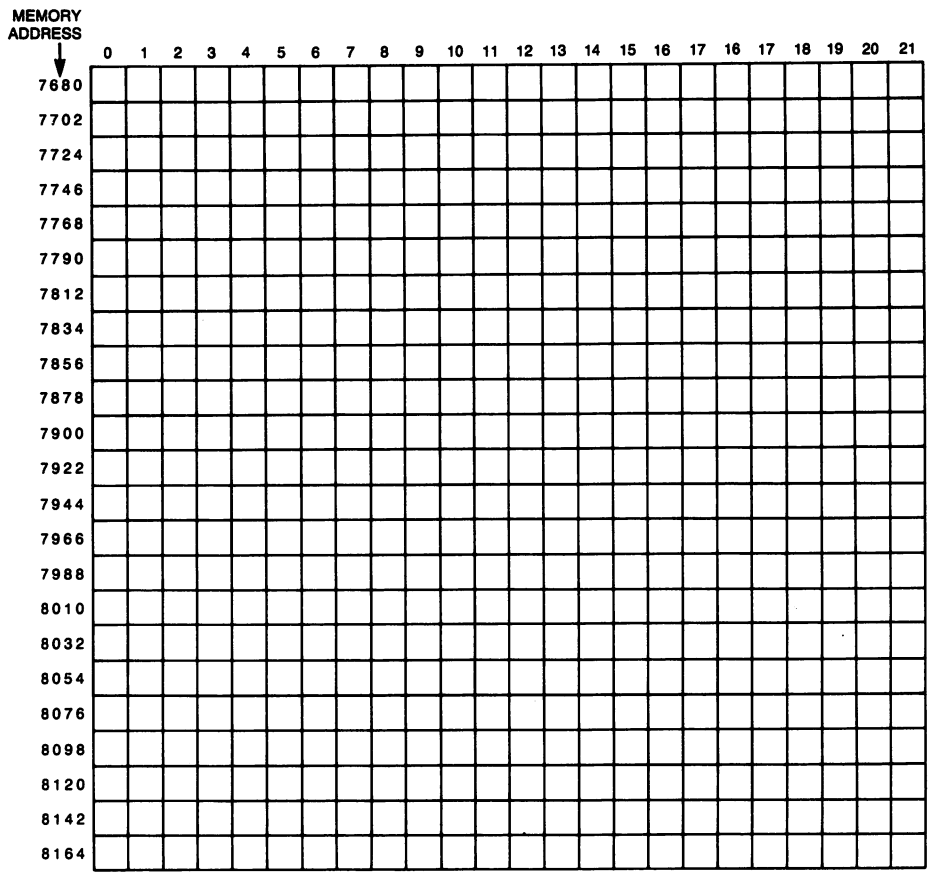



***Figure 5.18**–Grid layout of the video screen. The screen on the VIC-20 is 22 characters wide by 23 lines deep.*

yellow. To display the label in the correct color, we will fill the color array for the screen with the appropriate color number: green = 5, red = 2, and yellow = 7. The color memory grid looks exactly like the grid in Figure 5.18, except that all address numbers are offset by 30720.

In order to put the correct color into the color memory array, we must check to see if there was an alarm at the specified window or door. Figure 5.19 shows the software necessary to write the correct color labels to the screen of the VIC-20 computer. Figure 5.20 shows the complete program for controlling the home security system.

```
760 REM THIS WILL PUT IN LABELS OF CORRECT COLOR
770 RESTORE
780 FOR I = 1 TO 9
790   REM MAX NUMBER OF LINES TO CHECK
800   READ V1,V2
810   READ V3,V4
820   IF D(I) = 0   THEN Q1 = 5: REM NO ALARM
830   IF D(I) = 1   AND M(I) = 0 THEN Q1 = 2: REM ALARM
840   IF D(I) = 1   AND M(I) = 1 THEN Q1 = 7: REM ALARM BUT
      MASKED
850   POKE V2,V1
860   POKE V4,V3
870   POKE V2 + 30720,Q1
880   POKE V4 + 30720,Q1
890 NEXT I
900 REM WRITE W = WINDOWS, D = DOORS
910 FOR I = 8067 TO 8067 + 7
920   READ V1
930   POKE I,V1
940 NEXT I
950 FOR I = 8089 TO 8089 + 5
960   READ V1
970   POKE I,V1
980 NEXT I
990 REM FINISHED WITH A SINGLE PASS
1000 GOTO 560
1010 DATA 23,7684,49,7685
1020 DATA 23,7690,50,7691
1030 DATA 4,7698,49,7699
```

***Figure 5.19**—This routine will examine the D and M arrays to see if an alarm is present. Further, the code will print the labels in their correct colors, based on the alarm conditions (continues).*

```
1040 DATA 23,7877,51,7899
1050 DATA 23,7922,52,7944
1060 DATA 23,7979,53,7980
1070 DATA 4,7963,50,7985
1080 DATA 4,8084,51,8085
1090 DATA 23,8167,54,8168
1100 DATA 23,61,23,9,14,4,15,23
1110 DATA 4,61,4,15,15,18
1120 END
```

*Figure 5.19 (continued).*

```
 10 DIM A(5),D(16),M(16),A$(3)
 20 PRINT CHR$(147)
 30 FOR I = 1 TO 16
 40   D(I) = 0
 50   M(I) = 0
 60 NEXT I
 70 REM THE ABOVE ZEROED OUT ARRAYS
 80 PRINT "DO YOU WANT TO MASK ANY DOOR OR WINDOW?"
 90 INPUT A$
100 IF A$ = "N" THEN 180
110 FOR I = 1 TO 16
120   PRINT "DO YOU WANT TO MASK M(";I;")";
130   INPUT A$
140   IF A$ = "N" THEN 180
150   M(I) = 1
160 NEXT I
170 REM DRAW THE SCREEN NOW
180 PRINT CHR$(147)
190 REM SET COLOR OF EACH SCREEN CELL BLACK
200 FOR I = 38400 TO 38884 + 22
210   POKE I,0
```

***Figure 5.20**–Complete program for the home security system presented in this chapter (continues).*

```
220 NEXT I
230 REM TOP LINE OF SCREEN
240 FOR I = 7681 TO 7700
250   POKE 1,99
260 NEXT I
270 REM CORNERS
280 POKE 7680,79
290 POKE 7701,80
300 REM SIDES OF HOUSE
310 FOR I = 7702 TO 7944 STEP 22
320   POKE I,101
330   POKE I + 22,103
340 NEXT I
350 REM CORNER + SIDE
360 POKE 7966,101
370 POKE 7987,122
380 REM FIRST BOTTOM LINE
390 FOR I = 7976 TO 7980
400   POKE I,100
410 NEXT I
420 REM SMALLER SIDES
430 FOR I = 7988 TO 8142 STEP 22
440   POKE I,101
450   POKE I + 9,103
460 NEXT I
470 REM CORNERS
480 POKE 8164,76
490 POKE 8173,122
500 REM BOTTOM LINE
510 FOR I = 8165 TO 8172
520   POKE I,100
530 NEXT I
540 REM SCREEN IS NOW DRAWN, NO LABELS IN YET
550 REM NOW TO GET THE PEEK DATA
560 A(1) = PEEK (38912 + 2)
```

*Figure 5.20 (continued).*

```
570  A(2) = PEEK (38912 + 4)
580  T = 1
590  F = 1
600  GOSUB 670
610  T = 2
620  F = 9
630  GOSUB 670
640  GOTO 770
650  REM SUBROUTINE TO FILL DATA ARRAY
660  REM
670  R1 = 128
680  R2 = A(T)
690  FOR I = F TO F + 7
700    IF R2 - R1 < 0 THEN 730
710    R2 = R2 - R1
720    D(I) = 1
730    R1 = R1/2
740  NEXT I
750  RETURN
760  REM THIS WILL PUT IN LABELS OF CORRECT COLOR
770  RESTORE
780  FOR I = 1 TO 9
790    REM MAX NUMBER OF LINES TO CHECK
800    READ V1,V2
810    READ V3,V4
820    IF D(I) = 0 THEN Q1 = 5
830    IF D(I) = 1 AND M(I) = 0 THEN Q1 = 2
840    IF D(I) = 1 AND M(I) = 1 THEN Q1 = 7
850    POKE V2,V1
860    POKE V4,V3
870    POKE V2 + 30720,Q1
880    POKE V4 + 30720,Q1
890  NEXT I
900  REM WRITE W = WINDOWS, D = DOORS
910  FOR I = 8067 TO 8067 + 7
```

*Figure 5.20 (continued).*

```
 920    READ V1
 930    POKE I,V1
 940  NEXT I
 950  FOR I = 8089 TO 8089 + 5
 960    READ V1
 970    POKE I,V1
 980  NEXT I
 990  REM FINISHED WITH A SINGLE PASS
1000  GOTO 560
1010  DATA 23,7684,49,7685
1020  DATA 23,7690,50,7691
1030  DATA 4,7698,49,7699
1040  DATA 23,7877,51,7899
1050  DATA 23,7922,52,7944
1060  DATA 23,7979,53,7980
1070  DATA 4,7963,50,7985
1080  DATA 4,8084,51,8085
1090  DATA 23,8167,54,8168
1100  DATA 23,61,23,9,14,4,15,23
1110  DATA 4,61,4,15,15,18
1120  END
```

*Figure 5.20 (continued).*

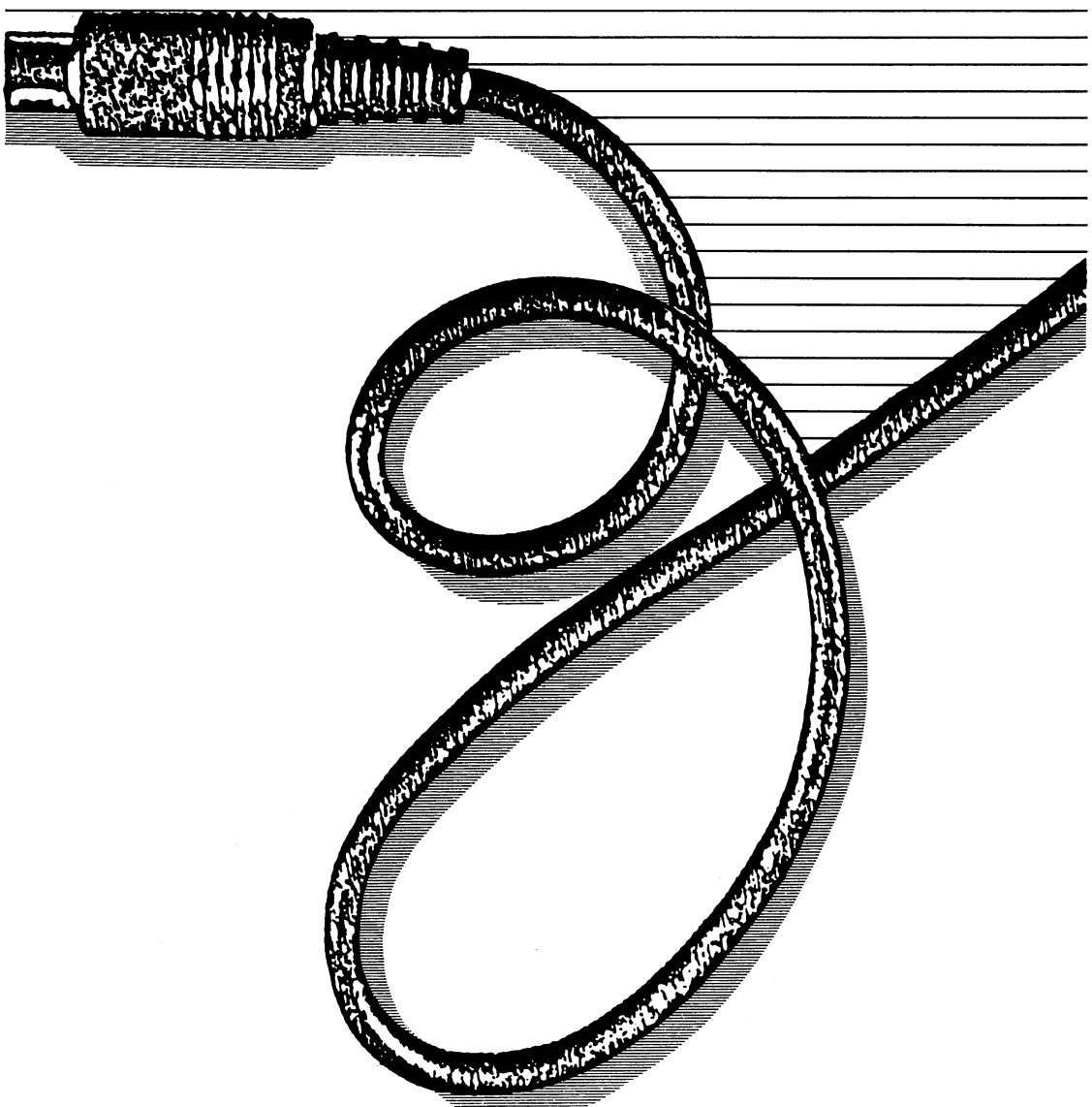
## 5.9 SUMMARY

In this chapter we have presented the complete design of a home security system using the VIC-20 computer as the system controller. The chapter started with a clear definition of the design. From there we discussed the hardware necessary to realize the interface, and after the hardware was shown, the software was presented.

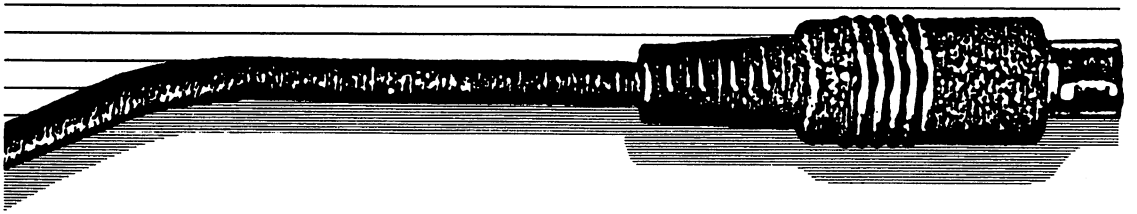
The details presented in this chapter are very typical of computer-controlled systems. That is, once the hardware interface is designed and connected, the problem becomes one of writing the software to control it.

We have also presented some guidelines for program development, which is a useful step when the external hardware is cumbersome to use or is not complete. The system shown in this chapter was designed to illustrate general principles, and its primary use is for instruction. However, if you can understand the major parts of this system, then construction of another, more elaborate system will be much easier.

# ADDING A VOICE TO THE VIC-20 COMPUTER



# 6



In this chapter we will discuss how to add a computer voice to the VIC computer. This topic falls under the broad category of "Voice Synthesis." We first show and discuss a general block diagram of a speech circuit. From there we present the hardware for a speech synthesizer. Do not be alarmed! The hardware for this circuit is extremely simple and easy to use with VIC-20.

After the hardware is discussed, we concentrate on the topic of writing software for voice control. We will give examples of programs written in BASIC for generating words, phrases, or commands. The speech synthesizer circuit will allow an unlimited vocabulary of words to be used. You will see for yourself just how easy it is to control the hardware that produces speech. Even if you are not experienced wth hardware, the circuit shown is very easy to construct. Once it is constructed, you will have a pleasant time making your system talk when you want and say what you want. What's more, this chapter will allow you to understand how speech synthesis can be used for robotics and machine control.

## 6.1 PHONEME SPEECH SYNTHESIS

The type of speech synthesis we will use in this chapter is called *phoneme speech synthesis. Phoneme* is defined as a language's smallest fundamental unit of sound. When we speak, each word spoken comprises a set of phonemes strung together. Each word consists of a few sounds that, when spoken in succession, produce the entire word.

Let us take an example of a word and show the different phonemes that combine to produce it. Of course, the exact sounds will depend on which area of the country you are from and what accent you may have in your speech. Using phonemes, you can tailor a particular word to suit your individual taste. The word we will use as an example is *baby.*

This word is made of four individual phonemes. The first of these phonemes is "B," which is sounded like the two letters "BI," with the I being a short I. Next the phoneme with the sound "AY" is used. This sound is like the long A sound in the word *made.* Third, the "BI" phoneme again, followed by the phoneme "EE." This is the long "E" sound, as in the word *eat.* By stringing these four sounds together (BI-AY-BI-EE) the complete word *baby* can be formed.

The preceding example simply described how different phonemes can be combined to pronounce the word *baby.* Any word can be broken down into its respective phonemes. To reproduce the word, we need only to output its phonemes in the correct sequence. The speed at which the phonemes are output will change the pitch of the word.

## 6.2 THE SET OF PHONEMES

Figure 6.1 shows a set of 64 phonemes that are available for the Votrax phoneme synthesizer chip, the device that we will use in generating a voice for our computer. Each item in Figure 6.1 consists of a hexadecimal code, a phoneme symbol, an output duration, and an example word. We will use all of the information shown in these four categories during this chapter. However, at this point we can concentrate on the phoneme symbol and the example word.

| Phoneme Code | Phoneme Symbol | Duration (ms) | Example Word | Phoneme Code | Phoneme Symbol | Duration (ms) | Example Word |
|---|---|---|---|---|---|---|---|
| ØØ | EH3 | 59 | jacket | 2Ø | A | 185 | day |
| Ø1 | EH2 | 71 | enlist | 21 | AY | 65 | day |
| Ø2 | EH1 | 121 | heavy | 22 | Y1 | 80 | yard |
| Ø3 | PAØ | 47 | no sound | 23 | UH3 | 47 | mission |
| Ø4 | DT | 47 | butter | 24 | AH | 250 | mop |
| Ø5 | A2 | 71 | made | 25 | P | 103 | past |
| Ø6 | A1 | 103 | made | 26 | O | 185 | cold |
| Ø7 | ZH | 90 | azure | 27 | I | 185 | pin |
| Ø8 | AH2 | 71 | honest | 28 | U | 185 | move |
| Ø9 | 13 | 55 | inhibit | 29 | Y | 103 | any |
| ØA | 12 | 80 | inhibit | 2A | T | 71 | tap |
| ØB | 11 | 121 | inhibit | 2B | R | 90 | red |
| ØC | M | 103 | mat | 2C | E | 185 | meet |
| ØD | N | 80 | sun | 2D | W | 80 | win |
| ØE | B | 71 | bag | 2E | AE | 185 | dad |
| ØF | V | 71 | van | 2F | AE1 | 103 | after |
| 1Ø | CH* | 71 | chip | 30 | AW2 | 90 | salty |
| 11 | SH | 121 | shop | 31 | UH2 | 71 | about |
| 12 | Z | 71 | zoo | 32 | UH1 | 103 | uncle |
| 13 | AW1 | 146 | lawful | 33 | UH | 185 | cup |
| 14 | NG | 121 | thing | 34 | 02 | 80 | for |
| 15 | AH1 | 146 | father | 35 | 01 | 121 | aboard |
| 16 | 001 | 103 | looking | 36 | IU | 59 | you |
| 17 | 00 | 185 | book | 37 | U1 | 90 | you |
| 18 | L | 103 | land | 38 | THV | 80 | the |
| 19 | K | 80 | trick | 39 | TH | 71 | thin |
| 1A | J* | 47 | judge | 3A | ER | 146 | bird |
| 1B | H | 71 | hello | 3B | EH | 185 | get |
| 1C | G | 71 | get | 3C | E1 | 121 | be |
| 1D | F | 103 | fast | 3D | AW | 253 | call |
| 1E | D | 55 | paid | 3E | PA1 | 185 | no sound |
| 1F | S | 90 | pass | 3F | STOP | 47 | no sound |

/T/ must precede /CH/ to produce CH sound.
/D/ must precede /J/ to produce J sound.

***Figure 6.1**–A list of the 64 phoneme codes that can be produced by the Votrax SC-01 chip. Reproduced by permission of Votrax, a division of Federal Screw Works.*

The example words shown in Figure 6.1 indicate how a phoneme will sound when generated electronically. Notice that there are phonemes listed in the table that produce no sound. These phonemes can be used for pauses between words. We will show examples of this concept as we proceed in the discussion.

## 6.3 HOW ARE THE CORRECT PHONEMES CHOSEN?

One of the most complicated aspects of phoneme speech synthesis (PSS) is choosing the correct phonemes for a particular word. There are usually reference tables that will help you get started. If you want to tailor the words to suit your individual taste, the selection of phonemes could become more difficult.

The best source for selecting the correct phonemes for a particular word is a ready-made list of words and their component phonemes, usually supplied by the manufacturer of the particular PSS chip you are using. Figure 6.2 shows two examples of phonemes supplied by Votrax for their chip. A more complete list of these words is given in Appendix E.

Another technique for selecting the correct phonemes, for words that are not included in the ready-made list, is to make your new word from "pieces" of ready-made words. This technique will usually get you "in the ball park." An example of this would be the following.

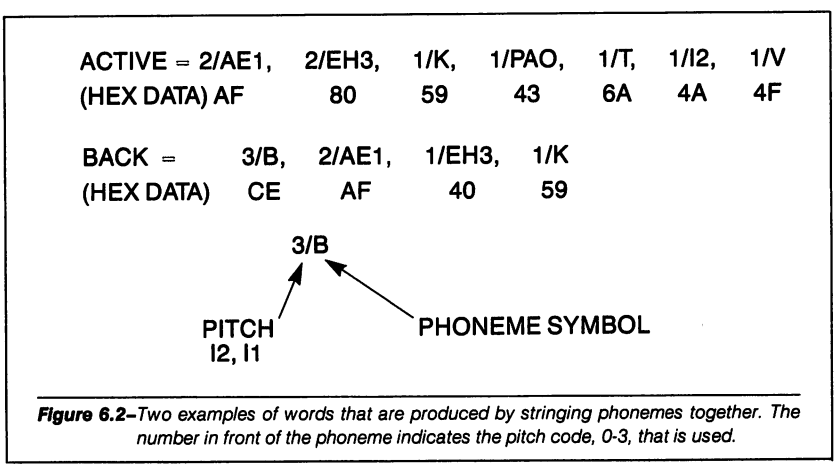



***Figure 6.2**–Two examples of words that are produced by stringing phonemes together. The number in front of the phoneme indicates the pitch code, 0-3, that is used.*

Suppose you wanted the phonemes for the word *Saturn*. This word is not included in the ready-made lists. We can make up this word from parts of the two words *Saturday* and *return*. The first part of *Saturday* and the latter part of *return* are used. Figure 6.3 shows the pieces of phoneme lists for *Saturday* and *return* that will be used to produce the phonemes for *Saturn*.

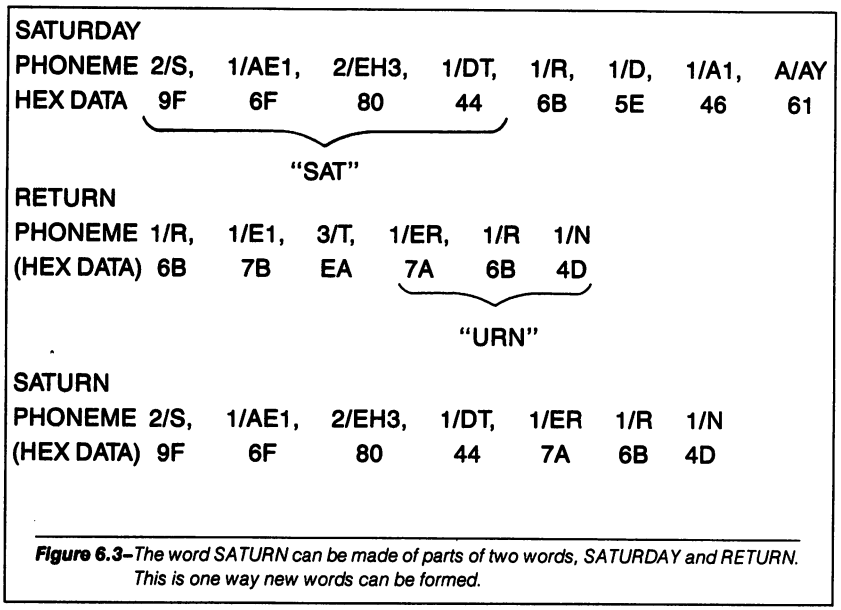



***Figure 6.3**–The word SATURN can be made of parts of two words, SATURDAY and RETURN. This is one way new words can be formed.*

## 6.4 VOTRAX SC-01 PHONEME SPEECH SYNTHESIZER

Up to this point we have discussed the concept of phoneme speech synthesis. We can now show how it can be realized. The main idea behind the realization is to electrically produce the selected phonemes in the correct order with the correct timing. Fortunately, the development of the SC-01 phoneme speech chip by Votrax has dramatically simplified this task. Figure 6.4 shows a flowchart of the events necessary to produce a word using phonemes.

Figure 6.5 shows a flow diagram and pinout of the Votrax SC-01 speech synthesizer chip. Let us review this flow diagram and show how the chip would be interfaced to a control device. We will discuss

the SC-01 chip at the user level only. The intent in this discussion is to allow you to use the SC-01 chip to generate phoneme speech. To accomplish that end we need not discuss in detail how the device operates internally.

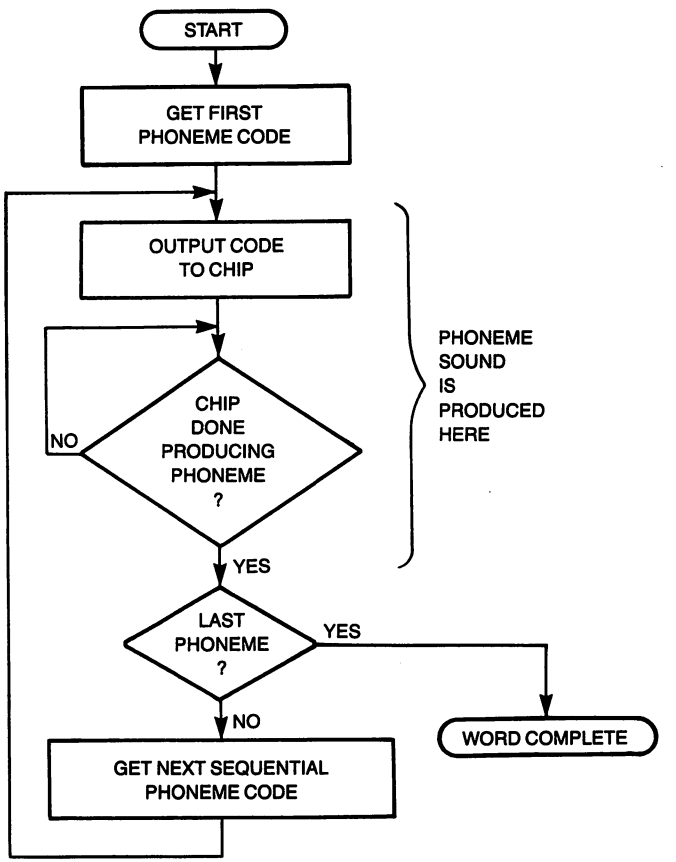
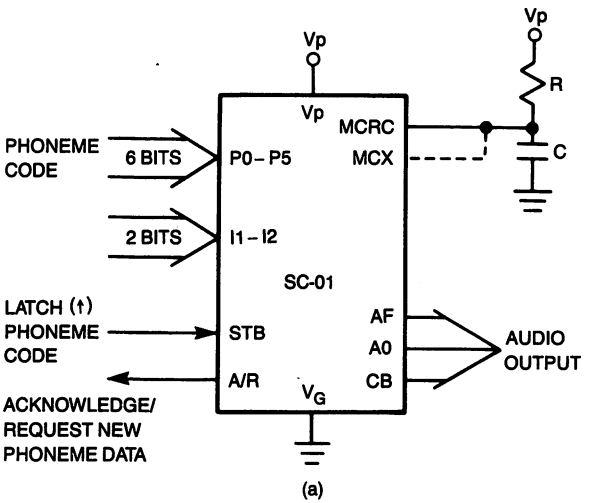
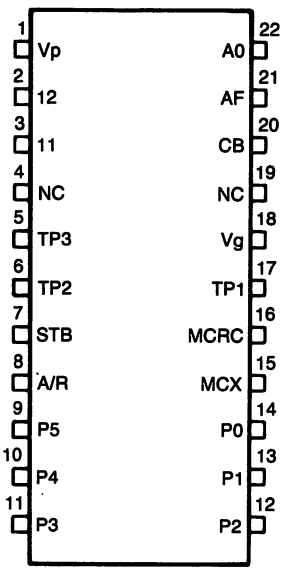



***Figure 6.4**–Flowchart of the steps necessary to produce speech with the SC-01 chip.*




NC = NO CONNECTION
TPX = NO CONNECTION
(b)

***Figure 6.5**–Pinout and flowchart of the SC-01 device.*

### Power Supply (VP,VG)

VP is the positive supply voltage applied to the device. The value is between 7 and 14 volts; normal operating voltage is equal to 12 volts. VG is connected to ground, or zero potential.

### Phoneme Code (P0 – P5)

These six input bits define the phoneme you wish the SC-01 to output. The SC-01 can output 64 different phonemes. Recall from Figure 6.1 the column labeled "Phoneme Code." This phoneme code represents the logical condition of the six input bits P0 – P5. In hexadecimal, the phoneme codes are from 00h to 3F. The computer that will control the SC-01 device supplies the phoneme code at the correct time. This will be discussed when we show the circuit for controlling the device. The term *hexadecimal* refers to a number in base 16. Our decimal number system is base 10. The digits in hexadecimal are 0–9,A,B,C,D,E,F. For this discussion you need only know that the phoneme codes are represented in hexadecimal notation. When the hexadecimal codes are used you will simply enter them into the computer as two keystrokes. For example, the hexadecimal code 0A would be entered as 0(keystroke) then A(keystroke).

### Pitch (I1,I2)

These two input bits determine the pitch of the selected phoneme, with 00 being the lowest pitch and 11 being the highest pitch. I1 is the *least* significant bit of the two.

### STB (Strobe for Phoneme Code)

This input signal is the strobe to the SC-01 that indicates when the external controller has applied the phoneme code and inflection bits to P0 – P5 and I1,I2. When the signal goes from a logical 0 to a logical 1, the SC-01 will operate on the applied inputs.

It should be noted that the phoneme code bits (P0 – P5) are latched with the strobe input. The I1,I2 bits, however, must be latched externally to the device. This is not a major drawback, and we will discuss

this further when the actual circuit for using the SC-01 is shown.

We further note that the STB input must remain low for a minimum time specified as 72 times the Master Clock Frequency. This means that at a clock frequency of 720 kilohertz, the period is equal to 1/720,000, or 1.40 microsecond. The STB must remain low for a time greater than 72×1.4 microseconds, or 100 microseconds. This time may not mean much to you, but it is an important consideration in designing the controller for the SC-01. We will use a circuit that meets this specification easily.

### A/R (Acknowledge/Request)

This output line is the handshake between the SC-01 and the control circuit. When a new phoneme code is strobed into the SC-01, signaled by the STB input going from a logical 0 to a logical 1, this output is set to a logical 0. After the SC-01 has processed the phoneme code, the A/R line is set to a logical 1, indicating that the device is electrically prepared to accept a new phoneme code.

The control hardware will monitor the logical level of this line to determine when a new phoneme is to be input to the device. We will give examples of how this line can be used in a *polled* or *interrupt* mode for a controlling microprocessor.

### MCRC (Master Clock Resistor-Capacitor)

This input determines the internal master clock frequency. Resistor-Capacitor (R-C) values are selected for an operating frequency of 720 Kilohertz. The frequency of the master clock is approximately equal to 1.25/RC. Note that the maximum resistance value is specified at 6.8 Kohms. When this R-C network is used as the clock frequency, the MCX input is connected to the MCRC input. If an external clock input is used, the MCRC input is connected to ground.

### MCX (Master Clock External)

This input allows an external clock to determine the operating frequency of the SC-01. If an external clock is not used, then this input is connected to the MCRC input line.

### AO (Audio Output)

This signal supplies the analog signal to an audio output device.

### AF (Audio Feedback)

This output signal is used with a class A or class B transistor audio amplifier.

### CB (Current Source for Class B)

This output is used as a current source for a Class B amplifier.

## 6.5 CONNECTING THE SC-01 TO THE VIC-20 PC

In this section we will discuss how the SC-01 can be connected to perform as a phoneme speech synthesizer. The circuit shown is very universal and can be connected to most home or business computers as well as any microprocessor-controlled system.

Figure 6.6 shows one way in which the SC-01 can be connected to the VIC-20 to produce speech. Note in this figure that we have shown a general connection to an output latch (a 74LS374) from some type of microprocessor system. This could be a personal computer or stand-alone microprocessor system. Let us now review Figure 6.6, highlighting the important points.

We first see in Figure 6.6 that the power supply, pin 1, or "VP," for the SC-01 chip is connected to +12 volts. The "VG" pin (18) of the device is connected to ground. A .1 microfarad capacitor is placed between the VP and VG pins. The oscillator RC is equal to 6.8 Kohms and approximately 330 picofarads.

The P0–P5, I1 and I2 pins are connected to output lines from a microprocessor-controlled system. The STB input is also connected to an output line from a microprocessor-controlled system. We note that the A/R output from the SC-01 will be connected back to the computer as an input to be monitored (polled), or as an external interrupt.

Audio output from the SC-01 is connected to a simple audio amplifier. The SC-01 output AO must be current-buffered prior to driving a

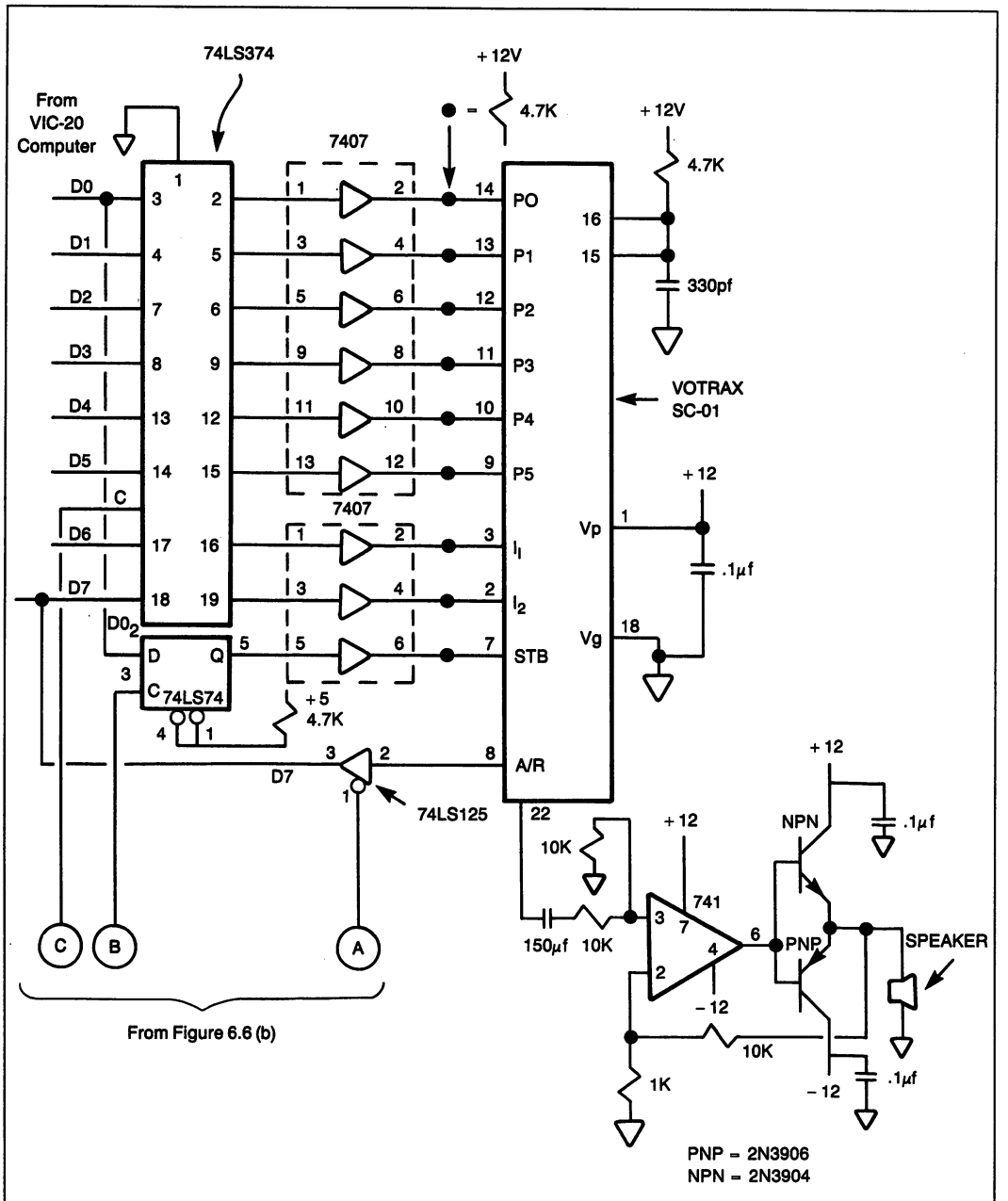



***Figure 6.6**–a, b) Complete schematic for connecting the SC-01 device to the VIC-20 computer (continues).*

speaker. This amplifier circuit will do the job nicely. However, if you have a favorite amplifier circuit, do not hesitate to experiment with it in place of the one shown.

Using the circuit shown in Figure 6.6, the SC-01 is capable of an unlimited vocabulary of speech. In the next section we will discuss the interface to the microprocessor-controlled system. Once this topic is covered, the discussion will focus on different software to control the SC-01.

## 6.6 CONTROLLING THE SC-01 WITH THE VIC-20 COMPUTER

As we begin this discussion, let us assume the P0–P5 and I1, I2 lines are connected to an output latch of the VIC-20. This is shown

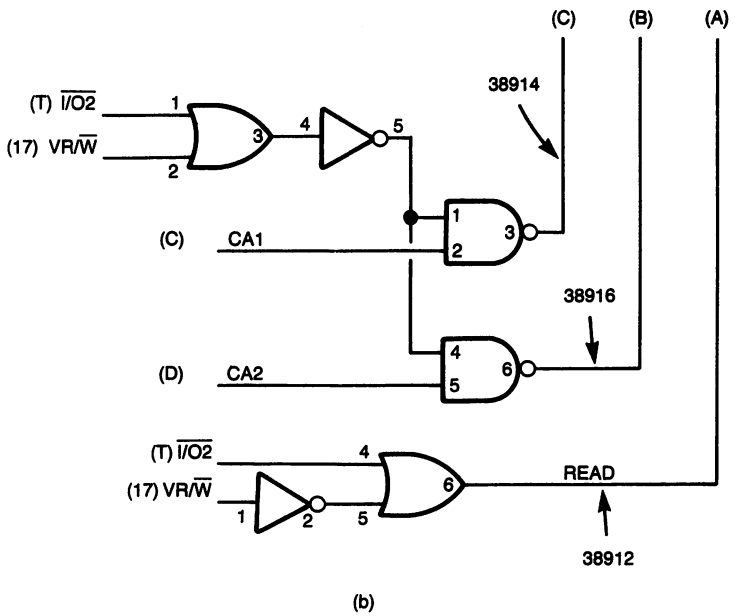



*Figure 6.6 (continued).*

in Figure 6.7. The STB line will also be connected to an output latch of the computer.

Let us concentrate on the A/R output line from the SC-01 device. This line is an input line to the VIC-20 computer.

With the A/R line connected in this configuration, the computer will electrically examine the logical state of the input. When the input line goes from a logical 0 to a logical 1, the next phoneme is input to the SC-01. The A/R output is set to a logical 0 when the STB line goes from a logical 0 to a logical 1.

Figure 6.8 shows a flowchart of the step that must be repeated to input phonemes to the SC-01 device with the A/R line connected as an input line to the computer. As each phoneme is input to the SC-01, the A/R line is electrically examined.

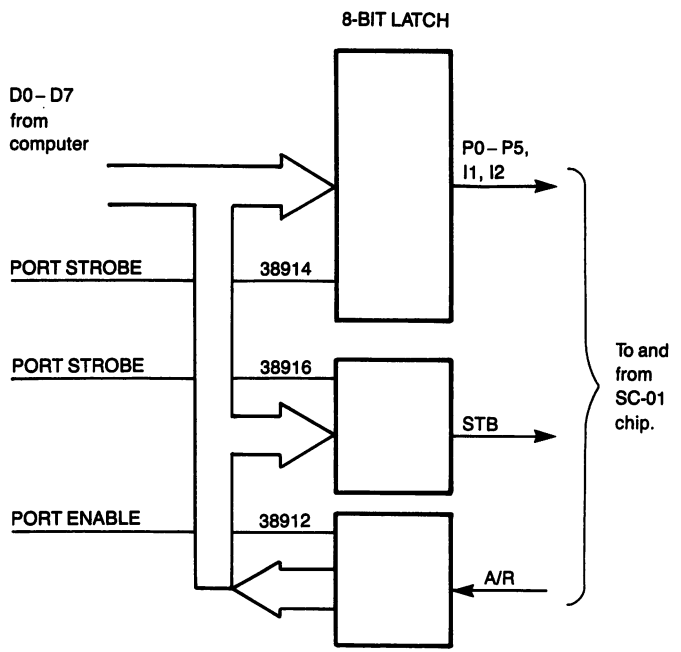



**Figure 6.7**–*Block diagram showing the addresses for communicating with the P0-P5, STB and A/R pins.*

## 6.7 EXAMPLE 1: OUTPUTTING A SINGLE PHONEME

In this section, we will write a program that will output a single phoneme code to the SC-01 device. With this program you will be able to hear exactly what each phoneme will sound like. Once this program is understood, we will begin stringing the phonemes together to produce some words.

The programming language will again be BASIC. This language was chosen because it is widely understood, and because the VIC-20 uses it. All programs will be well documented with comments, so you will clearly understand the function of each statement in the program. Figure 6.6 showed the schematic diagram of the electrical connection between the VIC-20 and the SC-01 device. It is important to note the POKE and PEEK addresses for the P0–P5, I1, I2, STB and A/R lines. These numbers are shown in Figure 6.7 as 38912 for the A/R line, 38914 for the P0-P5, I1, and I2 lines, and 38916 for the STB line.

The flowchart for this first program is shown in Figure 6.9. Let us discuss this flowchart and then write a BASIC program to realize the sequence of events.

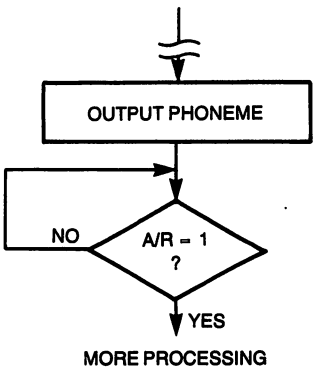



***Figure 6.8**–Flowchart of the sequence of events required to input phonemes to the SC-01 device.*

### Step 1: *Program lines 40 and 50*

The first event to occur in the program will be to input two hexadecimal digits. These digits will represent one of the 64 phoneme codes, with I1 and I2 set. In this step you can input your hexadecimal data as a two character string. That is, two keystrokes will be necessary to enter the two-character string.

### Step 2: *line 60*

In this step we will decide whether to terminate the program or not. If the data word you entered in Step 1 is equal to −1, then the flow

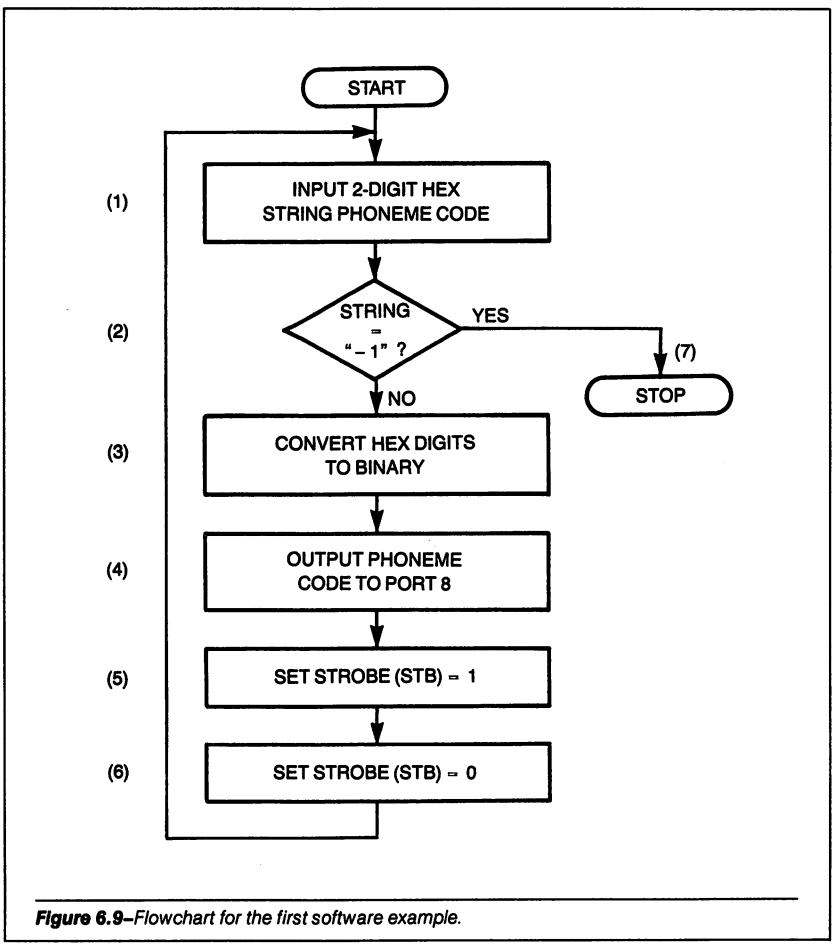



***Figure 6.9**–Flowchart for the first software example.*

of the program will proceed to Step 7 (STOP). If any other data was entered, then the flow will proceed to Step 3. Please note that you may use any symbol you desire to terminate your program; we chose – 1 simply as an example.

### Step 3: *lines 190–280*

We now convert the hexadecimal string data entered in Step 1 into 8 bits of binary data, to be output to the SC-01, using the subroutine shown in lines 190–280. In this subroutine, we convert each two-character string into a decimal equivalent, one at a time.

### Step 4: *line 110*

We are now ready to output the phoneme code to the SC-01 chip. The schematic diagram of Figure 6.6 showed that the output address of the phoneme code was equal to 38914. The code may be output using a POKE instruction.

### Step 5: *line 130*

The phoneme code is present on the P0– P5 and the I1 and I2 inputs to the SC-01 chip. We now apply the STB input. This step will set the STB input to the SC-01 to a logical 1. When this input goes to a logical 1, the SC-01 device starts to process the input data and output the desired phoneme. The A/R output line will go to a logical 0 when the STB output goes to a logical 1. The address of the STB output line is equal to 38916.

### Step 6: *line 140*

We now set the STB line to the SC-01 to a logical 0. Steps 5 and 6 are used together to apply a logical 1 pulse to the STB input line. When the SC-01 has finished processing the phoneme that was input, the A/R line will go to a logical 0. We do not need to examine the logical state of the A/R line in this program, because we are inputting only one phoneme at a time.

When the program has finished with Step 6, it will jump back to Step 1 and input another phoneme code from the keyboard. Figure 6.10 shows a BASIC program to realize the flowchart of Figure 6.9.

## 6.8 EXAMPLE 2: OUTPUTTING WORDS WITH THE SPEECH CHIP

In this example, we will output different complete words with the SC-01 device. The main idea is this: We will input the hexadecimal codes of all the phonemes required to synthesize a particular word. These hexadecimal codes will be stored in an array. The last byte of

```
 10 DIM T$(2)
 30 REM INPUT THE 2-DIGIT HEX STRING
 40 PRINT "INPUT THE HEX VALUE FOR EACH
    CHARACTER. -1 = END"
 50 INPUT T$
 60 IF T$ = "-1" THEN 290
 70 REM IF INPUT = -1 THEN STOP
 80 GOSUB 190
 90 REM SUBROUTINE TO CONVERT HEX TO DECIMAL
100 REM NOW TO OUTPUT PHONEME CODE TO THE SC-01
110 POKE 38914,T1
120 REM NOW TO PULSE THE STB INPUT TO THE SC-01
130 POKE 38916,1
140 POKE 38916,0
150 REM DO IT AGAIN
160 GOTO 40
170 REM THIS ROUTINE WILL CONVERT ASCII TO DECIMAL
180 REM EACH DIGIT IS CONVERTED ONE AT A TIME
190 C$ = MID$(T$,1,1)
200 M1 = 48
210 IF C$ >= "A" THEN M1 = 55
220 X = 16 * (ASC(C$) - M1)
230 C$ = MID$(T$,2,2)
240 M1 = 48
250 IF C$ >= "A" THEN M1 = 55
260 Y = ASC(C$) - M1
270 T1 = X + Y
280 RETURN
290 STOP
```

***Figure 6.10**–BASIC program to realize the flowchart of Figure 6.9.*

data in the array will again be – 1; this will be a signal that the input is complete.

When you enter the string "– 1" from the keyboard, the program will output all of the phonemes to the SC-01 chip in the order you entered them. If the phoneme codes were correct, then you will hear the word you entered being output from the chip over and over. It is best to let the chip repeat the word several times, so you can hear it. If the word is output only once, you may miss it. However, the number of times you need to hear it depends on how you input the word in the first place. Try it different ways to find the way you like it.

The flowchart for the program is shown in Figure 6.11. Let us discuss some of the important points of this flowchart. Some of the steps will not be discussed in detail, because we discussed them in the preceding example.

Steps 1 – 5 are used to input the phoneme codes that make up the entire word. When all phoneme codes are entered, you will enter a – 1. For example, the phoneme codes for the word *second* are 5F, 82, 59, 71, 4D, and 6A. You could enter these phonemes into the computer using this program, in the order shown. However, we will add two more codes, 3E and – 1. The 3E will pause with no sound for approximately 185 milliseconds, between repetitions of the word. The – 1 will logically inform the program that the last phoneme code has been entered. The complete data entry for this program to generate the word *second* will therefore be 5F, 82, 59, 71, 4D, 6A, 3E, and – 1. When the – 1 is entered, the program will jump to Step 6. The phoneme data will be output to the SC-01 in Steps 7 and 7a.

In Step 8 the logical level of the A/R output will be tested. If this output is a logical 0, the SC-01 has not finished processing the phoneme. In this case we will continue to loop on the test, waiting for the line to go to a logical 1.

When the A/R output equals a logical 1, the program will go to Step 10, where it will prepare to output the next phoneme. In Step 11 the program will test the output code for a – 1. If the output code does equal – 1, then the entire word has been output. In this case the program will go back to Step 6 and output the word again. If the code does not equal – 1, the program will go to Step 7 to output the next sequential phoneme code you had entered.

Figure 6.12 shows a complete BASIC program to realize the flowchart shown in Figure 6.11.

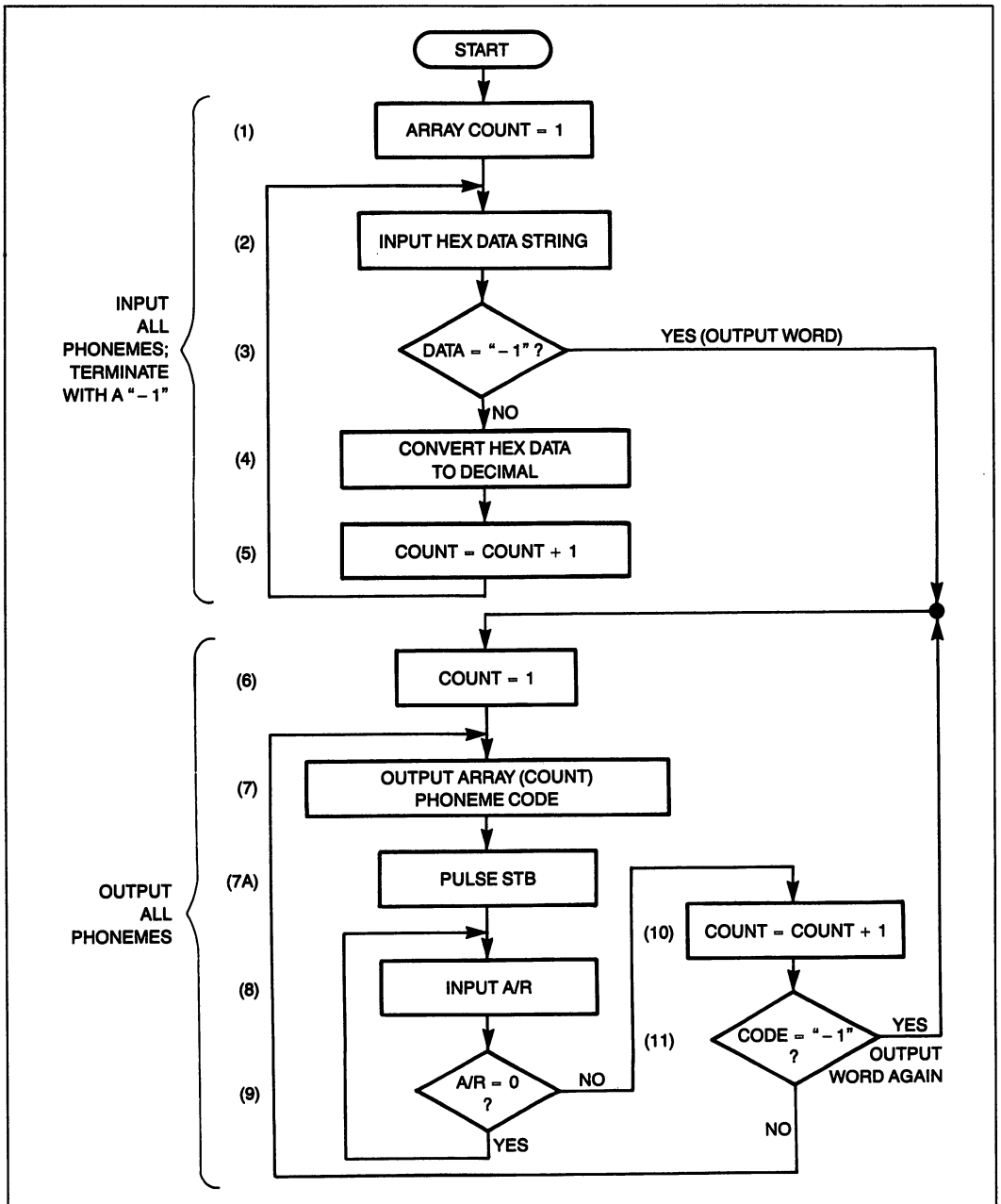



***Figure 6.11**–Flowchart for a program that will output the same word on the SC-01 chip repeatedly.*

```
10 DIM T$(2),V1(100)
20 C1 = 1
30 REM INPUT THE 2 DIGIT HEX STRING
40 PRINT "INPUT THE HEX VALUE FOR EACH
   CHARACTER - 1 = END"
50 INPUT T$
60 IF T$ = " - 1" THEN 110
70 REM IF INPUT = - 1 THEN END OF STRING
80 GOSUB 320
90 REM SUBROUTINE TO CONVERT HEX TO DECIMAL
100 GOTO 40
110 V1(C1) = - 1
120 REM SET LAST ELEMENT OF ARRAY EQUAL TO - 1
130 REM NOW TO OUTPUT THE PHONEME CODES IN THE ARRAY
140 C1 = 1
150 POKE 38914,V1(C1)
160 REM NOW TO PULSE THE STB INPUT TO THE SC-01
170 POKE 38916,1
180 POKE 38916,0
190 REM INPUT THE LOGICAL VALUE OF A/R
200 A = PEEK(38912)
210 REM A/R IS CONNECTED TO D7 INPUT,<128 = 0
220 IF A < 128 THEN 200
230 REM CHECK FOR LAST ELEMENT OF ARRAY
240 C1 = C1 + 1
250 IF V1(C1) = - 1 THEN C1 = 1
260 REM IF LAST ELEMENT, THEN START OVER IN ARRAY
270 REM IF NOT LAST ELEMENT, THEN OUTPUT NEXT ELEMENT
280 GOTO 150
290 REM SUBROUTINE FOR HEX TO DECIMAL
300 REM THIS ROUTINE WILL CONVERT ASCII TO DECIMAL
310 REM EACH DIGIT IS CONVERTED ONE AT A TIME
320 C$ = MID$(T$,1,1)
330 M1 = 48
340 IF C$ > = "A" THEN M1 = 55
350 X = 16 * (ASC(C$) - M1)
```

***Figure 6.12**–BASIC program to realize the flowchart of Figure 6.11 (continues).*

```
360  C$ = MID$(T$,2,2)
370  M1 = 48
380  IF C$ >= "A" THEN M1 = 55
390  Y = ASC(C$) - M1
400  T1 = X + Y
410  V1(C1) = T1
420  C1 = C1 + 1
430  RETURN
```

*Figure 6.12 (continued).*

## 6.9 EXAMPLE 3: OUTPUTTING A SENTENCE WITH THE SC-01

We have output single phonemes and words in the first two examples; now we are ready to output complete sentences or commands. This program will be very similar to the second example. Each word will have all of the phonemes input. Between words a 3E phoneme code will be entered; this will allow a pause between words. At the end of the sentence a – 1 will again signal that the string is completed.

Most sentences require many hexadecimal phoneme codes. Therefore, we will place the codes into DATA statements in BASIC, instead of inputting them during the program run, as we did before. For example, the command "TODAY IS SATURDAY" will require the phoneme codes shown below:

AA, A8, 5E, 46, 61, 3E, 67, 92, 3E, 9F, 6F, 80, 44, 6B, 5E, 46, 61, 3E, – 1

TODAY IS SATURDAY

In this phoneme-code string, notice the 3E between each word and the – 1 at the end of the sentence.

Figure 6.13 shows the flowchart for the program to output a sentence with the SC-01 synthesizer. Notice in Figure 6.13 that the data is treated as string data. The program will process the entire string, converting it to decimal data prior to outputting it to the SC-01. Figure 6.14 shows a BASIC program to realize the flowchart of Figure 6.13.

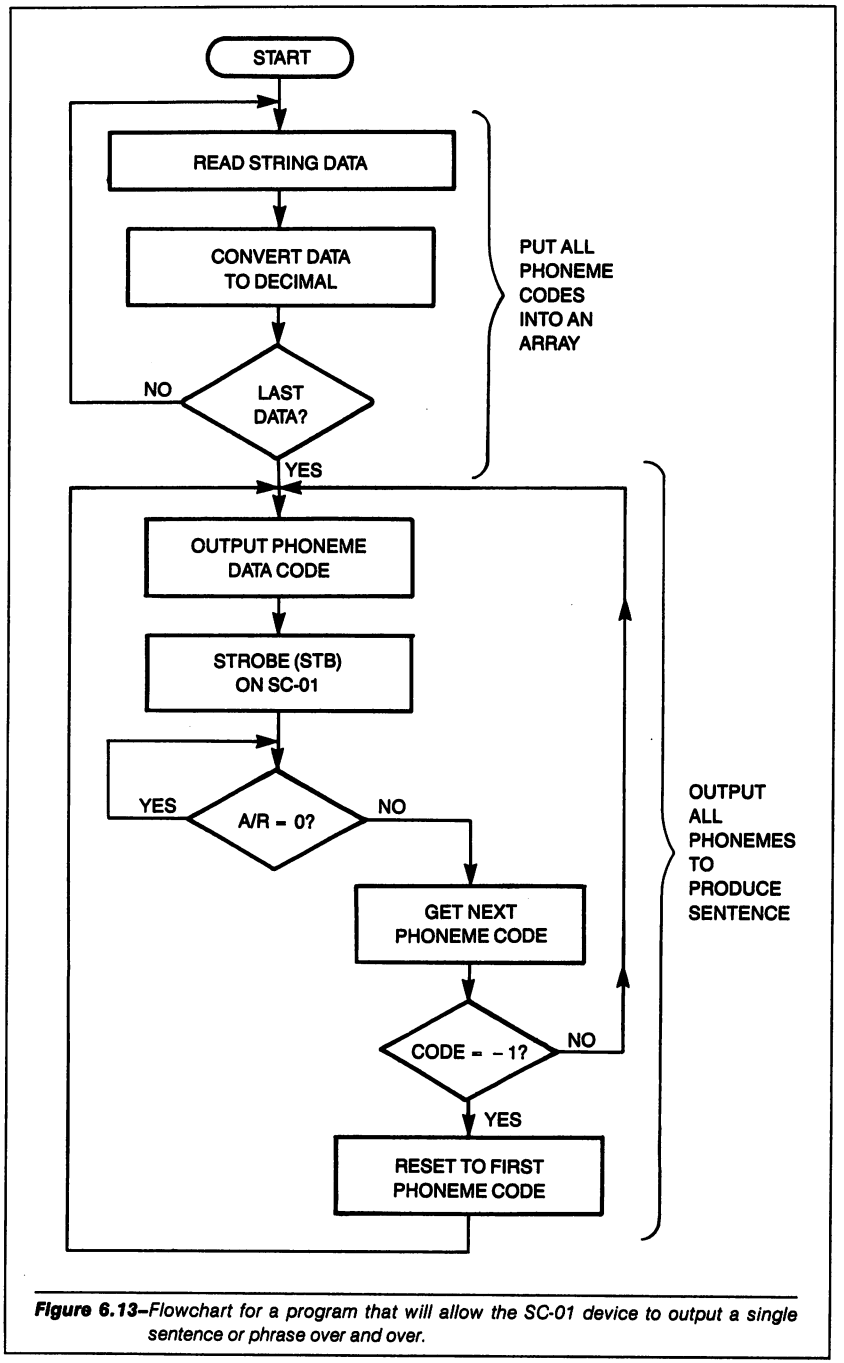



***Figure 6.13**–Flowchart for a program that will allow the SC-01 device to output a single sentence or phrase over and over.*

```
10  DIM T$(2),V1(300)
20  C1 = 1
30  RESTORE
40  REM CONVERT STRING DATA TO DECIMAL DATA AND PLACE IN ARRAY
50  READ T$
60  IF T$ = " – 1" THEN 100
70  GOSUB 290
80  REM SUBROUTINE WILL CONVERT STRING DATA TO DECIMAL
90  GOTO 50
100 V1(C1) = – 1
110 REM SET LAST VALUE IN ARRAY EQUAL TO – 1
120 C1 = 1
130 REM OUTPUT FIRST PHONEME CODE IN DATA LIST
140 POKE 38914,V1(C1)
150 REM PULSE THE STB INPUT TO THE SC-01
160 POKE 38916,1
170 POKE 38916,0
180 REM CHECK THE LOGICAL VALUE OF THE A/R OUTPUT
190 A = PEEK(38912)
200 IF A < 128 THEN 190
210 REM IF A/R = 0 THEN KEEP TESTING
220 C1 = C1 + 1
230 IF V1(C1) = – 1 THEN C1 = 1
240 REM CHECK FOR LAST ELEMENT OF THE ARRAY
250 REM IF NOT LAST ELEMENT THEN OUTPUT NEXT
260 REM IF LAST ELEMENT, RESET COUNT AND DO IT AGAIN
270 GOTO 140
280 REM SUBROUTINE FOR HEX TO DECIMAL
290 C$ = MID$(T$,1,1)
300 M1 = 48
310 IF C$ > = "A" THEN M1 = 55
320 X = 16 * (ASC(C$) – M1)
330 C$ = MID$(T$,2,2)
340 M1 = 48
350 IF C$ > = "A" THEN M1 = 55
```

***Figure 6.14**–BASIC program to realize the flowchart of Figure 6.13 (continues).*

```
360  Y = ASC(C$) - M1
370  T1 = X + Y
380  V1(C1) = T1
390  C1 = C1 + 1
400  RETURN
410  REM THIS IS THE DATA TO OUTPUT. 3E SEPARATES WORDS, -1 IS END
420  DATA AA,A8,5E,46,61,3E,67,92,3E,9F,6F,80,44,6B,5E,46,61,3E,-1
430  REM DATA IS "TODAY IS SATURDAY"
440  REM
```

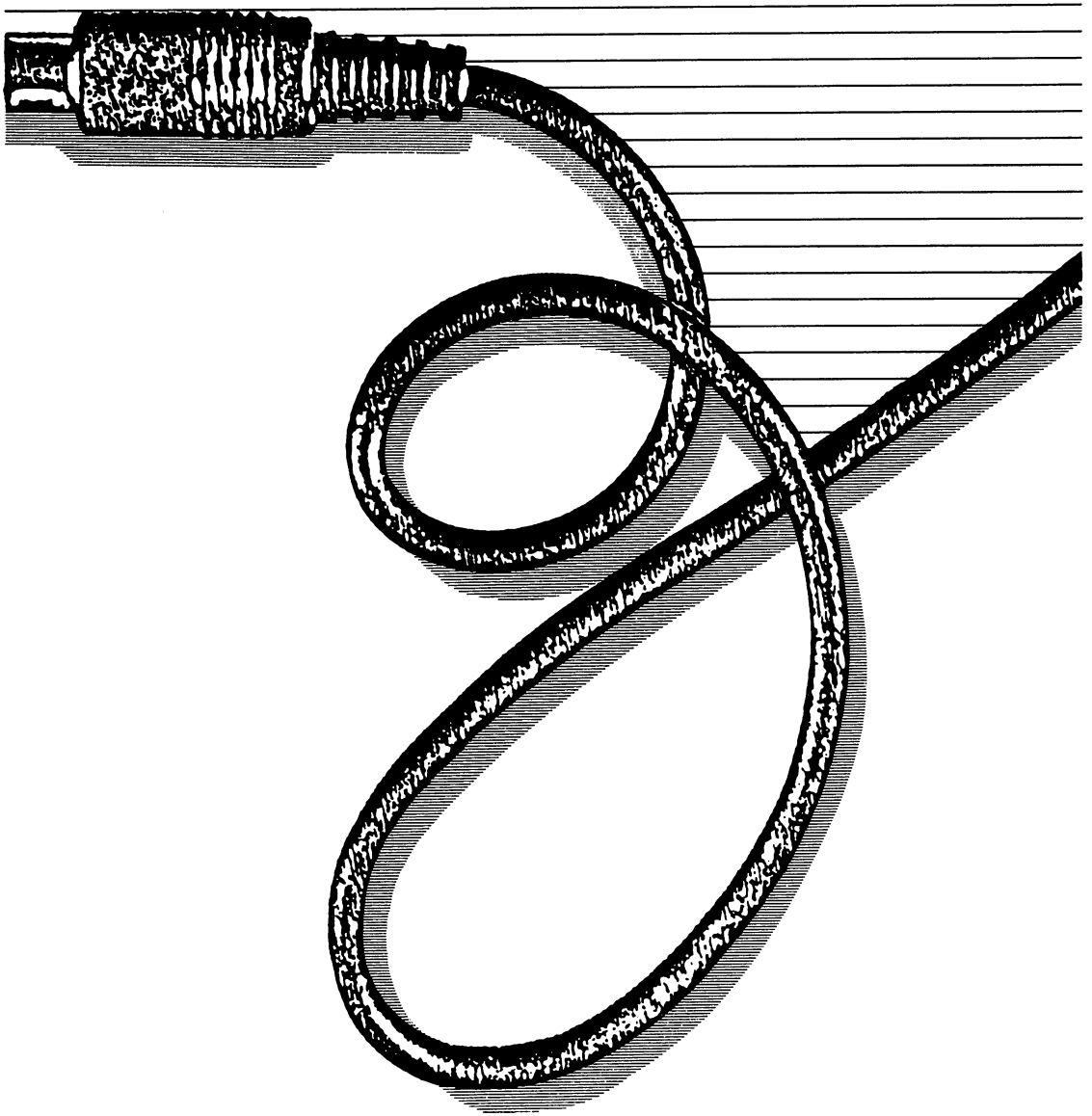
*Figure 6.14 (continued).*

## 6.10 SUMMARY

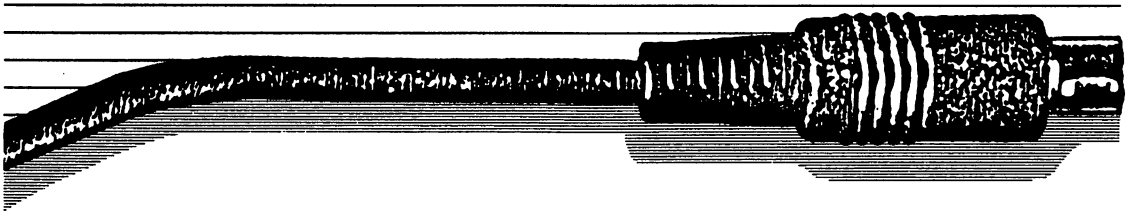
In this chapter we have discussed voice synthesis using the phoneme technique. This technique was chosen because it will allow you unlimited vocabulary that is easy to generate. We started by explaining how phoneme speech works, and then showed how to connect the SC-01 chip to a VIC-20 computer.

The chapter closed by giving three complete software examples of how the SC-01 chip can be used to generate vocabulary. If you would like to give your system a voice, phoneme speech synthesis may be employed quite easily. With a little experimentation, you will have your voice speaking whenever, and saying whatever, you desire.

# ANALOG VS. DIGITAL AND TRANSDUCERS



# 7



In this chapter we will examine two related concepts: 1) the difference between analog and digital events, and 2) the transducer. This discussion is intended to give the reader who is not familiar with these concepts a good understanding of them. If you are already familiar with the difference between analog and digital events, and with the concept of the transducer, you may want to skip this chapter.

However, it is important for anyone who wishes to learn about computer control to understand the difference between analog and digital events and the importance of transducers. As we will see, external devices are mostly analog in their operations, and some conversion is needed before they can operate under digital control.

## 7.1 ANALOG EVENTS

Briefly, an analog event is one whose output is variable. The following examples will illustrate this idea and demonstrate that we live in an analog world.

Nearly everyone is familiar with the common mercury thermometer, which, of course, measures temperature. A typical thermometer scale is shown in Figure 7.1. As we see, the temperature reading may vary anywhere on the scale: 98.6, 98.4, 98.2, 97.1, 100.3, 100.6, etc.

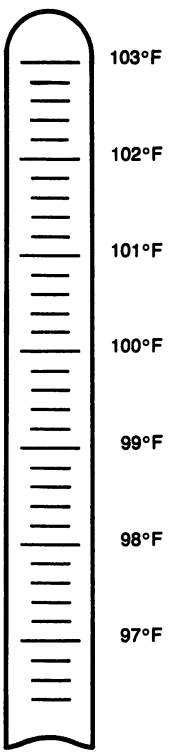



***Figure 7.1**–Typical thermometer showing the temperature scale in degrees Fahrenheit. The reading can be anywhere on the scale shown; there are an infinite number of potential output values on this continuous scale.*

The thermometer is an analog device because its output (the temperature reading) can have virtually any value. The range of values is limited by the scale on the thermometer (97 – 103° F), but within these limits there are an infinite number of possible readings. That is, the range is continuous. The point is that the temperature reading, like any analog output, can be *anywhere* between these limits.

A point of confusion may arise at this time. The term "digital" has been applied to thermometers that display numbers (digits) and use digital electronics in their display circuitry. In that sense (now the most common use of the term), they are indeed "digital" devices. As measuring instruments, however, these thermometers are analog devices, just as mercury thermometers are, because temperature is an analog phenomenon.

Sense perceptions may be thought of as analog events; consider the feeling of hunger. There are varying degrees of hunger, which seem to merge into each other. We can be full, almost full, a little bit hungry, very hungry, famished, or any number of different states in between, but we cannot tell where one state ends and another begins. Because the range of sensations is apparently continuous, hunger can be considered an analog event.

## 7.2 DIGITAL EVENTS

Digital events are not continuous but discrete. Their outputs are not infinitely variable within a range, but instead are limited to a finite number of predefined states. To illustrate this idea, let us re-examine our examples of analog events and consider what they would be like as digital events. In the two events we will describe, the number of discrete states or output values will be limited to two. We could have specified more values, but we have chosen only two, because that is the number of states possible in the binary logic used in digital electronics.

First, let's examine the thermometer. If this device were limited in the way we have described, it would have only two temperature output readings. The scale would appear as it does in Figure 7.2, where the temperature reading is either 70° F or 80° F. These two numbers were chosen at random to illustrate the point. A thermometer such as

this would be quite useless to us, because temperature is an analog event. It cannot be measured using a digital scale, because it varies. It would be a strange world indeed if there were only two possible values for temperature.

Now let's examine hunger as if it were a digital event, again limited to the two possible states we have specified. You would either feel full or hungry. This might not seem so bad, but the change from one to the other would be immediate, and it would be absolute. One second you would feel "stuffed," the next you would feel "famished." Fortunately, hunger is an analog event and actually varies in imperceptible stages.

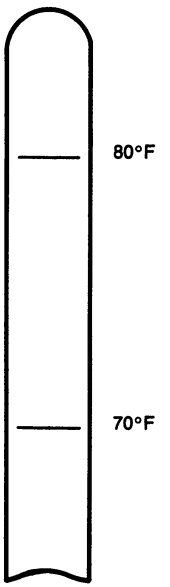



***Figure 7.2**–A thermometer with a binary scale. The temperature reading can only be 70° F or 80°F.*

The preceding examples were intended to show that a digital event has discrete output values. The contrast between digital and analog phenomena can also be illustrated by comparing a staircase and a ramp. The staircase represents digital phenomena and the ramp represents analog. On a staircase we must ascend or descend by fixed, predefined steps. On a ramp we can move any amount we desire. The closer the stairs are to one another, the smaller the steps we proceed by, and the closer we can come to some desired level. However, as the size of the steps decreases, the number of steps required to reach the same level must increase. If there were an infinite number of stairs in the staircase, we would have an analog event again. We will return to this analogy in Chapter 9 (see Figure 9.7).

## 7.3 PURELY DIGITAL EVENTS

Although most events in the real world are analog, some common digital events occur. Let's examine some.

One of the most common digital events is the turning on and off of a light. The light output has only two possible values: completely on, or completely off. (Of course, if you have dimmer switches installed in your home, the light output will vary, because you are using an analog device, a rheostat.)

A home furnace fan in a forced-air heating system is another digital event. Either the fan is on, running at a fixed, constant RPM, or it is off, not spinning at all. With a thermostat (another analog device) we can vary the temperature at which the fan will turn on. However, when it does turn on it will always run at the same speed. A further example of a digital, binary device is the ON-OFF switch used in television sets and home computers. The switch has two positions, ON or OFF. Television channel selectors and fans with more than one speed, however, are digital but not binary. There are a finite number of possible states, but more than two.

## 7.4 ANALOG AND DIGITAL ELECTRONICS

So far in this chapter, we have illustrated the difference between analog and digital events with examples from everyday life. Let us

now examine what the terms *analog* and *digital* mean when applied to electronics. In electronics, the terms refer to electrical quantities, such as resistance, current, and voltage. The quantity you may be most familiar with is voltage. For example, the output of a car battery is an analog voltage. A battery rated at 12 volts may actually produce 12.02 V, 12.1 V, 11.9 V, 12.6 V or any value near 12 volts. As the battery ages, the voltage output may drop: 11.9 V, 10.88 V, 10.73 V. The voltage output of the battery is analog, not digital, because it varies. A digital voltage output would have a finite number of possible values. A battery would have to produce exactly 12.0 V, 12.5 V, or 12.75 V. A battery does not meet these conditions; it is an analog device.

Home computers are called digital because there are a finite number of possible voltage levels for the electrical information being processed within the system. We saw what these levels were when discussing input and output in Chapters 2 and 3 of this text; there were only two possible voltage levels in the system. All digital circuits used in computers have this characteristic, but they may not use the same actual voltage levels to process information. A power supply (battery) of different voltages may be used to run the digital system, but the digital circuits will output and input data at only two different voltages. (See Figure 7.3.)

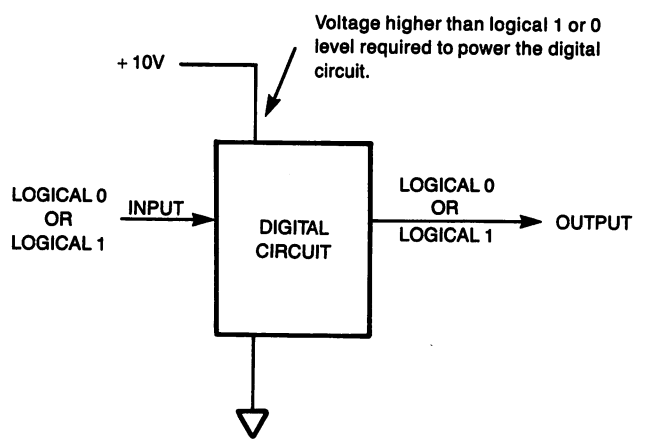



***Figure 7.3**–A block diagram showing that non-logical-level voltages may be required to power the digital circuits. However, the digital circuit will only process input and output at the logical 0 or logical 1 voltage levels.*

In a digital electronic circuit, the two possible output levels are called "logical 1" and "logical 0." Because there are only two states, digital circuits are designed to follow the logic of binary (or Boolean) mathematics, a number system that has only two digits, 0 and 1. (We normally work in the decimal number system, which has ten digits, 0–9.)

Home computers and personal computers are digital, binary machines. They will operate on digital information only. Most of the events that occur in the physical world are analog. If we wish to control and monitor these analog events with a digital computer, there is a conflict. For a graphic illustration of this conflict, see Figure 7.4. If you can understand the difference between analog and digital events, you will learn to resolve the incompatibility between them more easily.

## 7.5 TRANSDUCERS

In the remaining chapters of this text we will discuss how to bridge the gap between the analog events typical of the real world and the

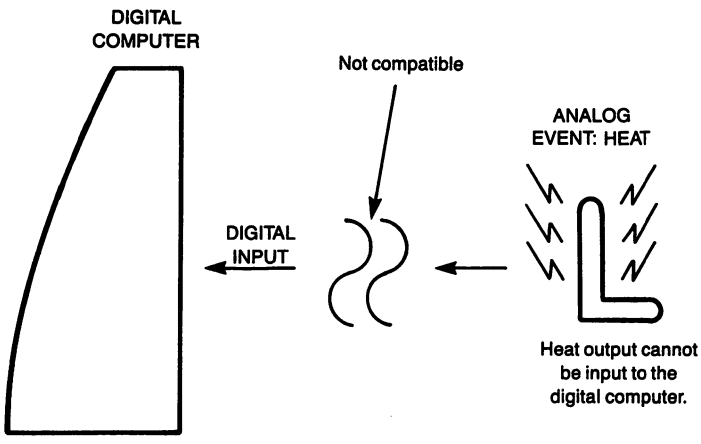



***Figure 7.4**–An analog event, such as heat or temperature, is not directly compatible with a digital computer; it cannot be measured or monitored without some intermediate device.*

digital electronics used to control or monitor them. Before we can begin to explore this problem, another piece of the puzzle must be explained. That remaining piece is the *transducer.*

We introduced the transducer in Chapter 1 and mentioned it again in Chapter 5. We return to it here to discuss this device in the context of analog-to-digital conversion. At this point you probably have a better understanding of how the transducer fits into the computer control problem, and of what such a device should do.

It has been stated that most of the phenomena we wish to monitor and control are analog, or continuous. That is, they can be measured anywhere on a continuous scale of values. However, these values represent different physical characteristics. For example, the various scales of temperature measure the physical form of energy, heat. There are an infinite number of different temperature readings, but the form of energy they all represent is heat. Pressure is another physical quality, another form of energy, that we sometimes wish to monitor with the digital computer. It is also an analog event, because there are an infinite number of pressure outputs.

What we need is some way to transform the various different physical forms of energy into one that can be used in an electronic environment. That new form should be some electrical quantity: voltage, current, resistance, capacitance, or inductance. After the physical form of energy has been changed into electricity, it can be further processed using electronic techniques. Finally, we will need to input the energy, processed into electronic information, into a digital computer. The entire process is shown in the block diagram of Figure 7.5.

The device that transforms the physical quantity into an electrical quantity is called a *transducer.* Transducers are generally classified according to the physical form of energy that they transform into an electrical quantity. For example, a transducer that will transform pressure into an electrical form is called a pressure transducer. One that will transform temperature into an equivalent electrical quantity is called a temperature transducer, and so on. Often, however, the qualifying term, "pressure" or "temperature," is omitted in the literature, since it is generally known what type of energy has to be measured. The reader must infer the type of transducer from the surrounding text.

In Chapter 8 we will describe a temperature transducer and use it to enable the computer to measure an analog quantity, temperature.

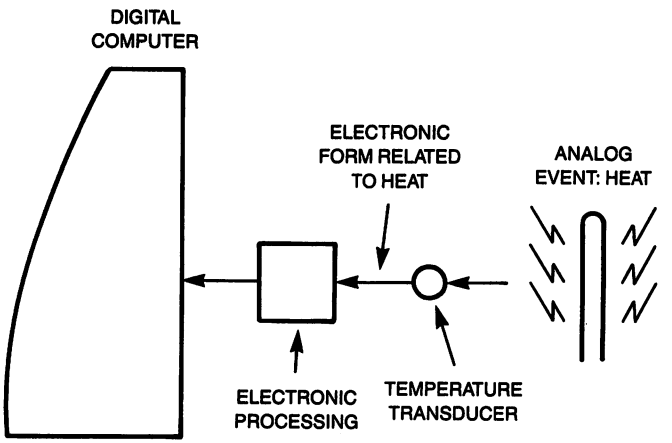


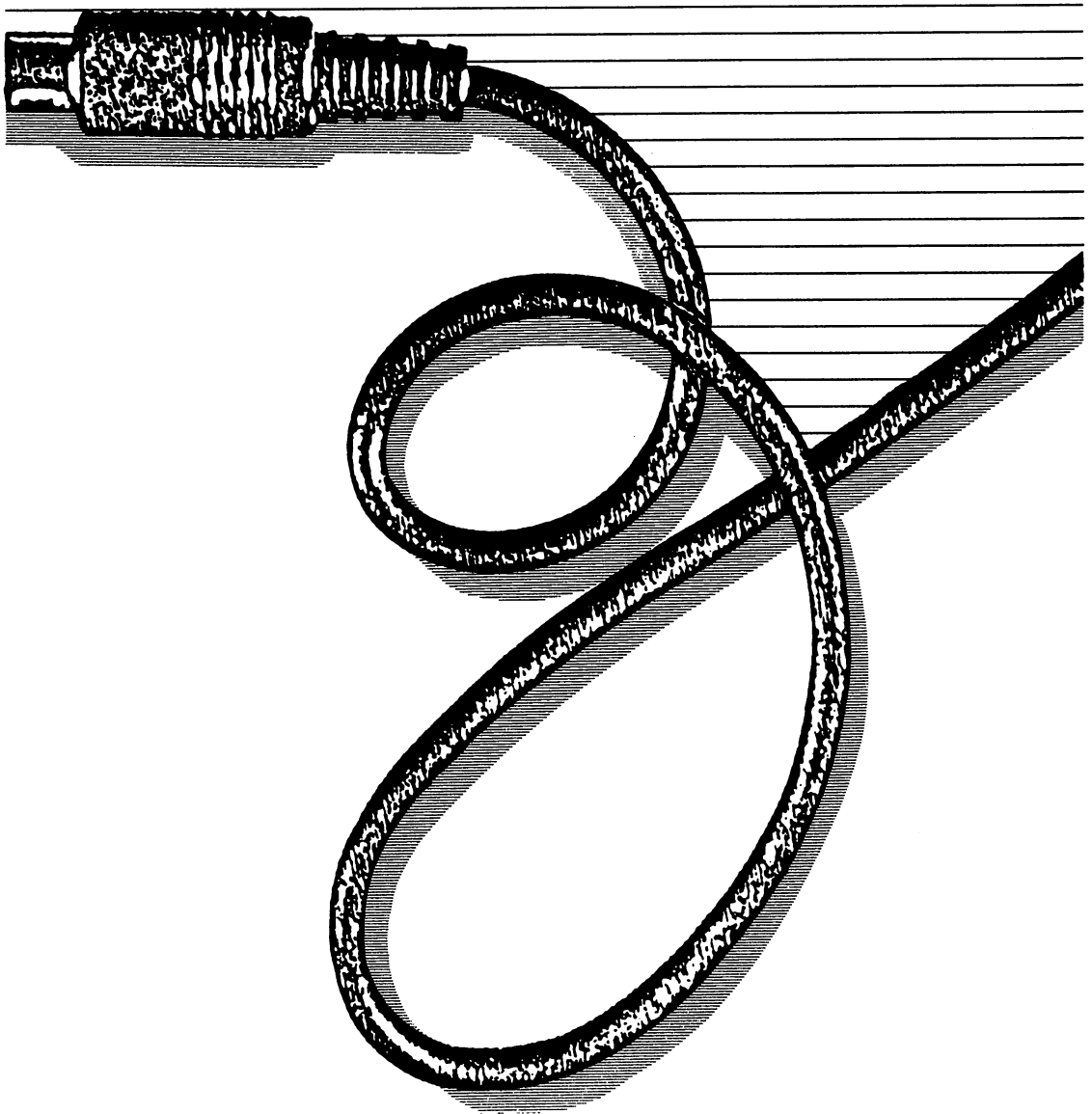

***Figure 7.5**—The flow of information in the monitoring system is from the analog event, heat, into the physical transducer. The transducer will transform the temperature output into its electrical equivalent. The electrical signal can then be processed electronically and, finally, input to the digital computer.*

## 7.6 SUMMARY

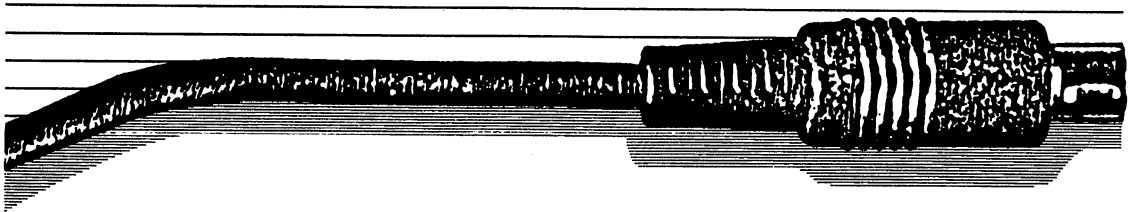
In this chapter we have discussed the meaning of the terms *analog* and *digital,* illustrating them with common events that occur in the world. After this introduction, we applied these concepts to electronics, and discussed digital and analog electrical quantities. Finally, we saw what a *transducer* is, and discussed how it fits into an automatic computer control environment.

When we undertake the job of designing or using a computer-controlled system, we will need to know the difference between analog and digital quantities, and how each of the external devices being controlled or monitored outputs information. Sometimes the output is analog, and sometimes it is digital. If the external device produces an analog output, some type of conversion will be necessary. In the following chapter, we will discuss analog-to-digital conversion, and show how to input a physical quantity, temperature, to the VIC-20 PC. In doing so, we will use information presented in this chapter.

# ANALOG-TO-DIGITAL CONVERSION FOR THE VIC-20 COMPUTER



# 8



In this chapter we will show how to use the VIC-20 Personal Computer to monitor any external instrument that outputs an analog voltage. The discussion starts off by showing a block diagram of the overall concept involved in this type of computer monitoring. From there, the discussion focuses on how an analog-to-digital converter operates. The discussion will be basic enough to allow the beginner to understand the principles of analog-to-digital conversion.

After the analog-to-digital converter is presented, this chapter will show how to electrically connect an analog-to-digital converter to the VIC-20 via the expansion connector slot. Finally, a complete system for monitoring temperature will be explained. At the conclusion of this chapter, you will have a good understanding of how devices that output analog voltages may be monitored using a digital computer like the VIC-20.

## 8.1 BLOCK DIAGRAM OF THE PROBLEM

Figure 8.1 shows a block diagram of the overall concept that we must use. In this diagram there are three major blocks:

1. Block 1 is the digital monitoring device. In this case it will be the VIC-20.
2. Block 2 will transform the analog voltage output from the external device into an equivalent digital word.
3. Block 3 is the external device itself. This device is a transducer of some sort. The transducer will output a voltage that has an amplitude dependent on the physical event being monitored.

In the last chapter we discussed what was meant by an analog event. In review, an analog event is one that can be measured at an infinite number of possible output levels (within a given range). An example was pressure. Pressure can be measured at any value from a few pounds per square inch (psi) to many hundreds of psi. Further, the value can be any number of psi between these two limits.

If the event to be monitored by the digital computer is analog, a transducer capable of accurately reporting the analog changes is necessary. For example, as the pressure being measured changes, the

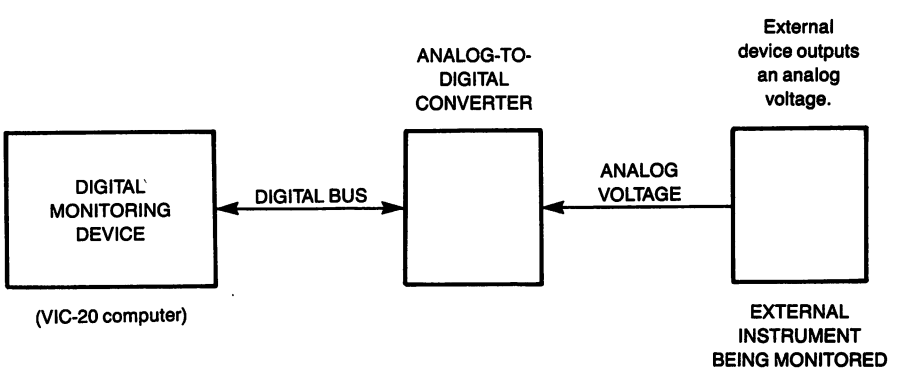



***Figure 8.1**–Block diagram showing an overall view of the concepts involved in monitoring an analog voltage with a digital system.*

transducer must reflect this change in its output. Whenever the pressure changes a little, either increasing or decreasing, the voltage output from the pressure transducer must increase or decrease in proportion. (See Figure 8.2.)

These changes are in contrast to a "digital" transducer. The switches we used in Chapter 5 for the home security system are an example of such a transducer. We needed these transducers to indicate whether the windows and doors were open or closed. Although it is true that a window or door can be wide open or just cracked open, in the example of Chapter 5 we did not need to distinguish between the degrees of openness; the event we were monitoring was "digital." That is, for our purposes, the door or window was either open or closed.

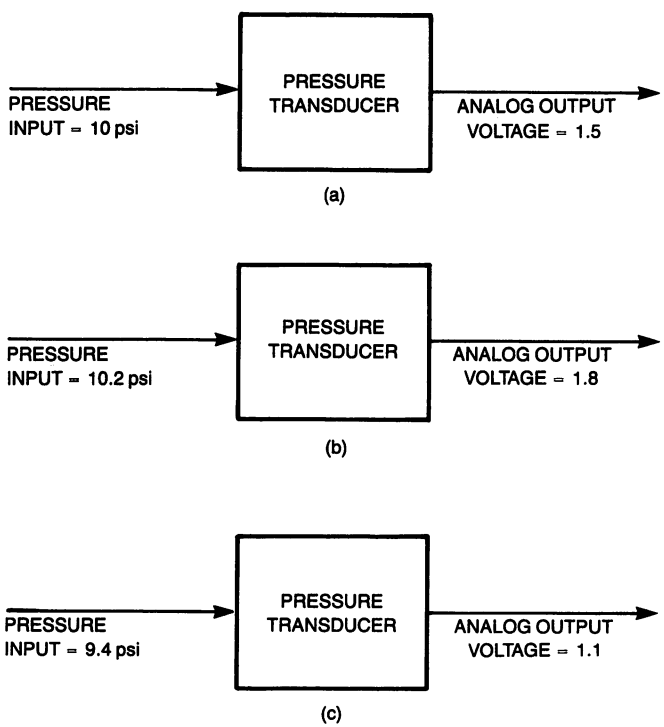



***Figure 8.2**–a,b,c) Each of these block diagrams shows the analog voltage output from a given pressure input. Note that as the pressure changes at the input of the transducer, the analog voltage must change in proportion.*

The transducers we are concerned with now are analog output transducers. This means the transducer's output will vary in voltage anywhere from a low voltage, 0.0 volts, to a high voltage, approximately 5.0 volts. We use this voltage range simply as an example. Typically, the outputs of transducers are variable from any negative voltage to any positive voltage. Voltage variations depend on the type of transducer used. An example of an actual analog voltage transducer will be presented later in this chapter.

For the present, let us assume that an analog transducer will output a certain range of voltage. This analog voltage will be input to the second block of Figure 8.1, labeled "analog-to-digital converter." The digital output of the analog-to-digital converter is then input to the VIC-20 block.

The VIC-20 will electrically input digital information being output from the analog-to-digital converter. As we will see, this digital information will be equivalent to the analog voltage that is input to the analog-to-digital converter.

## 8.2 THE ANALOG-TO-DIGITAL CONVERTER

Let us now define exactly what is meant by the term "analog-to-digital conversion," and discuss how we can make use of this principle. In general terms, the function of analog-to-digital conversion is to transform (or convert) an analog input voltage into the digital output word that is its mathematical representation.

This function must be performed before a digital computer can input the analog voltage. A device that performs this type of function is called an *analog-to-digital converter* or simply ADC. The remainder of this section will be devoted to explaining how a typical ADC operates. After this discussion, we will present some examples of how the ADC device operates in a digital system environment.

We can begin the discussion by looking at Figure 8.3, a block diagram of a typical ADC. This diagram shows general ADC inputs and outputs. Let us concentrate on the function of each input and output shown in Figure 8.3, beginning with the power inputs. The ADC consists of electronic circuits, which require a source of Direct Current (DC) power. The power input lines are shown at the top of Figure 8.3.

Typical power connections will be + and −12 or 15 volts, +5 volts, and ground.

The + and −12-to-15 volt DC supplies are used to power some of the internal circuits, such as operational amplifiers and voltage comparators, located inside the ADC package. A +5 volt supply will usually power the digital electronics inside the ADC package. Data sheets on any particular ADC will specify the DC power required for proper electrical operation.

In Figure 8.3 we also see the input line labeled "ANALOG VOLTAGE." This is the analog voltage that will be transformed into its equivalent digital word. Any analog-to-digital converter has a specified range of analog voltages it can accept as input. Typical ADC analog input voltage ranges are 0 to +5 volts, −5 to +5 volts, −10 to +10 volts, and 0 to +10 volts. These are just a few of the many different voltage ranges that are commonly available. Refer to the manufacturer's data sheet for exact specifications on the analog input voltage range for a particular ADC.

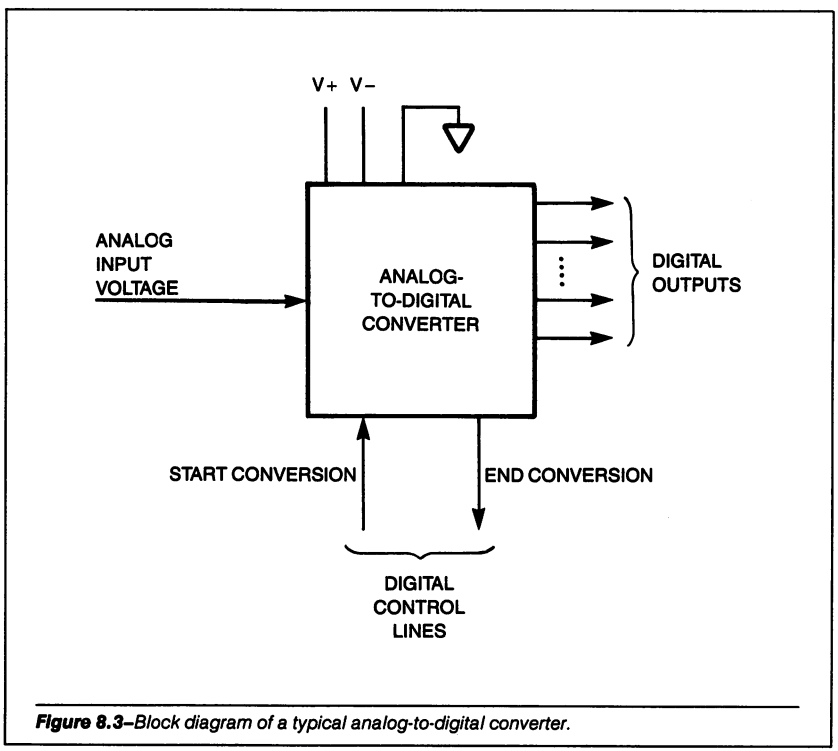



***Figure 8.3**–Block diagram of a typical analog-to-digital converter.*

Some of the input voltage ranges listed are both negative (−) and positive (+), while some are only positive. If the input voltage to the ADC can range from positive to negative, the ADC is said to be *bipolar,* meaning that it has two electrical poles. If the input voltage range is from zero to a positive value, the ADC is said to be *unipolar,* meaning that it has only one pole. The ADC we will use later in the chapter is unipolar.

The next section of Figure 8.3 to examine shows eight digital output lines. The number of digital output lines on an ADC will vary. Typical numbers of output lines are 6, 8, 10, 12 and 16. In general, the greater the number of output lines, the more expensive the ADC.

The digital output lines on the ADC will be set to a logical 1 or a logical 0 as determined by the analog input voltage. Later in the discussion, it will be shown how to calculate which digital outputs will be a logical 1 and which will be a logical 0. For now, think of the digital output lines as the physical connections to the digital monitoring system. In our case, the connections are to the VIC-20.

Finally, in Figure 8.3, we see two digital control lines, labeled "START CONVERSION" and "END CONVERSION." Each line has a unique function. The "START CONVERSION" input control line is a digital signal that will electrically inform the ADC to start converting the analog input to a digital output word. This signal is generated by the digital monitoring device (in this case, the VIC-20). When the VIC-20 is electrically prepared to accept the digital output word from the ADC, the "START CONVERSION" input control line to the device is *asserted;* that is, the input signal will be placed into the logical state that will start the action. Since that logical state could be either a 1 or 0, the term "asserted" is used to generalize.

The "END CONVERSION" output line shown in Figure 8.3 will be used to electrically inform the digital monitoring device that an analog-to-digital conversion is complete. Notice that the ADC is itself a small electronic system. It will take a short time for the ADC to adjust the digital outputs to new values based on the analog input voltage. The exact time required will vary from ADC to ADC, but typical times are between 10 and 200 microseconds, or .000010 and .000200 seconds.

When the computer electrically requests an analog-to-digital conversion, it must electrically monitor the "END OF CONVERSION" output line during the conversion. This monitoring will electrically

determine when the conversion is complete. Figure 8.4 shows a flowchart of how a computer electrically communicates with an ADC.

In Figure 8.4 we see that the analog-to-digital conversion will be started when the VIC-20 asserts the "START CONVERSION" input line to the ADC. Once the hardware is connected, this assertion will be accomplished using software. After the conversion is started, the VIC-20 will electrically monitor the "END OF CONVERSION" line from the ADC. When this line is true, the VIC-20 will read (input) the digital output lines of the ADC. This flowchart will be realized later in this chapter when the ADC is electrically connected to the VIC-20.

You have probably noticed that the discussion thus far has not gone

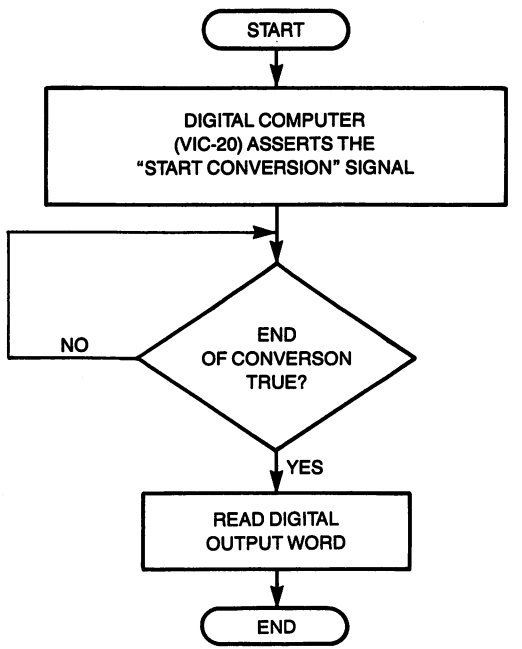



***Figure 8.4**–Flowchart showing the sequence of events required for electrical communication with an ADC.*

into the details of *how* the ADC internally performs the analog-to-digital conversion. That topic is beyond the scope of this text. We present the ADC at the user's level, treating it as a "black box" that will perform a certain function in a certain prescribed manner. This is precisely the way ADCs are used. All of the electronics that actually perform the conversion are usually out of the user's control or access. This chapter, therefore, simply discusses how to make the ADC perform its prescribed function. Understanding this introduction to the ADC inputs and outputs has been our first step toward using this type of device with the VIC-20 computer.

## 8.3 CALCULATING THE DIGITAL OUTPUTS OF THE ADC

In this section we will discuss how to determine the relationship between the logical conditions of the digital outputs and the voltage level of the analog input. For this discussion we will use an ADC that is unipolar. The acceptable voltage input range will be from 0.0 volts to + 10.0 volts. There will be eight digital output lines. A block diagram of this type of device was shown in Figure 8.3.

The digital outputs will be labeled D0 – D7. These outputs will be assigned the same numerical weights we have given to bit positions in data bytes elsewhere in this text. A table of the digital output lines and the corresponding weights is shown in Figure 8.5.

| Digital Output Line | Weight |
|---|---|
| D0 | 1 |
| D1 | 2 |
| D2 | 4 |
| D3 | 8 |
| D4 | 16 |
| D5 | 32 |
| D6 | 64 |
| D7 | 128 |

***Figure 8.5**–The ADC's output lines have the same numerical weights as the VIC-20 data bus lines.*

As we have seen before, the minimum weight at the digital outputs will occur when all lines are a logical 0, and the maximum weight at the digital outputs will occur when all are a logical 1. The minimum and maximum weights are 0 and 255, respectively.

The minimum analog voltage input to the ADC is 0.0 volts. The maximum analog input voltage to the ADC is 10.0 volts. If we relate the maximum and minimum input voltage to the maximum and minimum digital output weights, we have:

0.0 volts input = weight 0 output

10.0 volts input = weight 255 output

All input voltages between 0 and 10 volts will have a corresponding digital weight between 0 and 255.

There are 256 different weights available for the input voltages, because there are eight digital outputs. The number of different weights available with a particular ADC is calculated by the following equation:

number of digital weights = 2 raised to the power of (number of digital output lines)

In our case we had eight digital output lines, so the number of digital weights was equal to $2^8$, or 256. If we had an ADC with 10 digital output lines, the number of digital weights would equal $2^{10}$, or 1024.

Since we know how many unique digital weights are available for the analog input voltage range, we can divide that range into equal increments. In our example we have 255 available digital weights. (One weight must be subtracted because of the 0 at the start.) Therefore, the range between 0 and 10 volts will be divided into 255 equal pieces by the following equation:

$$\frac{\text{voltage range}}{255} = .03921569$$

Using this information, we can generate a list of input voltages and their corresponding digital output weights. This list is shown in Figure 8.6; it should be read *across*, rather than down.

We see in Figure 8.6 that the equivalent input voltage increases in discrete steps of approximately .04 volts. For example, suppose an input voltage to the ADC is equal to 1.5 volts. There is no digital output weight exactly equivalent to 1.5 volts, so from the list of Figure 8.6

we must choose an output weight that will yield the voltage closest to 1.5 volts. We see that 1.5 volts is between 1.49 volts (a weight of 38) and 1.53 volts (a weight of 39). The weight of the ADC digital outputs for an input voltage of 1.5 volts is closest to 38, so we can use that number. However, it is not an exact representation of the analog input voltage.

The point is that the ADC can only digitally represent analog voltages between 0 and 10 volts to a resolution of approximately .04 volts. This error is dependent on the number of digital output lines and the analog voltage input range. If the value of .04 volts is too great a range, then a different ADC, with more digital output lines and a finer resolution, must be chosen.

When using the ADC, we do not want to have to look up the equivalent voltage each time a conversion takes place. It would be much easier if the voltage could be calculated, based on the digital output word. This would lend itself well to computer control. These calculations can be accomplished by the following equation: The analog voltage is equal to the digital output weight multiplied by the resolution of the ADC.

**Digital Weights and Voltages**

| D | V | D | V | D | V | D | V | D | V |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00 | 1 | 0.04 | 2 | 0.08 | 3 | 0.12 | 4 | 0.16 |
| 5 | 0.20 | 6 | 0.24 | 7 | 0.27 | 8 | 0.31 | 9 | 0.35 |
| 10 | 0.39 | 11 | 0.43 | 12 | 0.47 | 13 | 0.51 | 14 | 0.55 |
| 15 | 0.59 | 16 | 0.63 | 17 | 0.67 | 18 | 0.71 | 19 | 0.75 |
| 20 | 0.78 | 21 | 0.82 | 22 | 0.86 | 23 | 0.90 | 24 | 0.94 |
| 25 | 0.98 | 26 | 1.02 | 27 | 1.06 | 28 | 1.10 | 29 | 1.14 |
| 30 | 1.18 | 31 | 1.22 | 32 | 1.25 | 33 | 1.29 | 34 | 1.33 |
| 35 | 1.37 | 36 | 1.41 | 37 | 1.45 | 38 | 1.49 | 39 | 1.53 |
| 40 | 1.57 | 41 | 1.61 | 42 | 1.65 | 43 | 1.69 | 44 | 1.73 |
| 45 | 1.76 | 46 | 1.80 | 47 | 1.84 | 48 | 1.88 | 49 | 1.92 |
| 50 | 1.96 | 51 | 2.00 | 52 | 2.04 | 53 | 2.08 | 54 | 2.12 |
| 55 | 2.16 | 56 | 2.20 | 57 | 2.24 | 58 | 2.27 | 59 | 2.31 |
| 60 | 2.35 | 61 | 2.39 | 62 | 2.43 | 63 | 2.47 | 64 | 2.51 |
| 65 | 2.55 | 66 | 2.59 | 67 | 2.63 | 68 | 2.67 | 69 | 2.71 |
| 70 | 2.75 | 71 | 2.78 | 72 | 2.82 | 73 | 2.86 | 74 | 2.90 |

D = DIGITAL WEIGHT
V = VOLTAGE

***Figure 8.6**–List of digital ouput weights and corresponding analog input voltages they represent (continues).*

**Digital Weights and Voltages**

| D | V | D | V | D | V | D | V | D | V |
|---|---|---|---|---|---|---|---|---|---|
| 75 | 2.94 | 76 | 2.98 | 77 | 3.02 | 78 | 3.06 | 79 | 3.10 |
| 80 | 3.14 | 81 | 3.18 | 82 | 3.22 | 83 | 3.25 | 84 | 3.29 |
| 85 | 3.33 | 86 | 3.37 | 87 | 3.41 | 88 | 3.45 | 89 | 3.49 |
| 90 | 3.53 | 91 | 3.57 | 92 | 3.61 | 93 | 3.65 | 94 | 3.69 |
| 95 | 3.73 | 96 | 3.76 | 97 | 3.80 | 98 | 3.84 | 99 | 3.88 |
| 100 | 3.91 | 101 | 3.96 | 102 | 4.00 | 103 | 4.04 | 104 | 4.08 |
| 105 | 4.12 | 106 | 4.16 | 107 | 4.20 | 108 | 4.24 | 109 | 4.27 |
| 110 | 4.31 | 111 | 4.35 | 112 | 4.39 | 113 | 4.43 | 114 | 4.47 |
| 115 | 4.51 | 116 | 4.55 | 117 | 4.59 | 118 | 4.63 | 119 | 4.67 |
| 120 | 4.71 | 121 | 4.75 | 122 | 4.78 | 123 | 4.82 | 124 | 4.86 |
| 125 | 4.90 | 126 | 4.94 | 127 | 4.98 | 128 | 5.02 | 129 | 5.06 |
| 130 | 5.10 | 131 | 5.14 | 132 | 5.18 | 133 | 5.22 | 134 | 5.25 |
| 135 | 5.29 | 136 | 5.33 | 137 | 5.37 | 138 | 5.41 | 139 | 5.45 |
| 140 | 5.49 | 141 | 5.53 | 142 | 5.57 | 143 | 5.61 | 144 | 5.65 |
| 145 | 5.69 | 146 | 5.73 | 147 | 5.76 | 148 | 5.80 | 149 | 5.84 |
| 150 | 5.88 | 151 | 5.92 | 152 | 5.96 | 153 | 6.00 | 154 | 6.04 |
| 155 | 6.08 | 156 | 6.12 | 157 | 6.16 | 158 | 6.20 | 159 | 6.24 |
| 160 | 6.27 | 161 | 6.31 | 162 | 6.35 | 163 | 6.39 | 164 | 6.43 |
| 165 | 6.47 | 166 | 6.51 | 167 | 6.55 | 168 | 6.59 | 169 | 6.63 |
| 170 | 6.67 | 171 | 6.71 | 172 | 6.75 | 173 | 6.78 | 174 | 6.82 |
| 175 | 6.86 | 176 | 6.90 | 177 | 6.94 | 178 | 6.98 | 179 | 7.02 |
| 180 | 7.06 | 181 | 7.10 | 182 | 7.14 | 183 | 7.18 | 184 | 7.22 |
| 185 | 7.25 | 186 | 7.29 | 187 | 7.33 | 188 | 7.37 | 189 | 7.41 |
| 190 | 7.45 | 191 | 7.49 | 192 | 7.53 | 193 | 7.57 | 194 | 7.61 |
| 195 | 7.65 | 196 | 7.69 | 197 | 7.73 | 198 | 7.76 | 199 | 7.80 |
| 200 | 7.84 | 201 | 7.88 | 202 | 7.92 | 203 | 7.96 | 204 | 8.00 |
| 205 | 8.04 | 206 | 8.08 | 207 | 8.12 | 208 | 8.16 | 209 | 8.20 |
| 210 | 8.24 | 211 | 8.27 | 212 | 8.31 | 213 | 8.35 | 214 | 8.39 |
| 215 | 8.43 | 216 | 8.47 | 217 | 8.51 | 218 | 8.55 | 219 | 8.59 |
| 220 | 8.63 | 221 | 8.67 | 222 | 8.71 | 223 | 8.75 | 224 | 8.78 |
| 225 | 8.82 | 226 | 8.86 | 227 | 8.90 | 228 | 8.94 | 229 | 8.98 |
| 230 | 9.02 | 231 | 9.06 | 232 | 9.10 | 233 | 9.14 | 234 | 9.18 |
| 235 | 9.22 | 236 | 9.25 | 237 | 9.29 | 238 | 9.33 | 239 | 9.37 |
| 240 | 9.41 | 241 | 9.45 | 242 | 9.49 | 243 | 9.53 | 244 | 9.57 |
| 245 | 9.61 | 246 | 9.65 | 247 | 9.69 | 248 | 9.73 | 249 | 9.76 |
| 250 | 9.80 | 251 | 9.84 | 252 | 9.88 | 253 | 9.92 | 254 | 9.96 |
| 255 | 10.00 | | | | | | | | |

D = DIGITAL WEIGHT
V = VOLTAGE

***Figure 8.6 (continued).***

analog voltage = digital weight * ( 10/255 )

For example, suppose the weight read from the ADC was equal to 125. This would equal an input voltage of:

125 * (10/255) = 4.9 volts

Remember that this voltage may actually equal 4.9 volts or slightly less than 4.94 volts, because of the resolution of the ADC.

In its present form, our equation is dependent on the input voltage range and the number of digital output lines on the ADC. These values must be modified in order for the equation to fit a particular application.

## 8.4 CONNECTING THE ADC TO THE VIC-20 PERSONAL COMPUTER

Figure 8.7 shows a complete schematic for connecting an actual ADC, Analog Devices' model AD570, to the VIC-20. In this section we will discuss exactly how each part of Figure 8.7 operates. The AD570 is very similar in its operating characteristics to the general example we gave in Figure 8.3. A data sheet for this device is included in Appendix A.

The analog-to-digital converter is labeled IC6 in Figure 8.7. The device is powered by −12-volt and +5-volt lines, which must be output from an external source, as the VIC-20 only outputs +5 volts on the expansion connector.

On the right side of the ADC (IC6) is the analog input voltage to be converted. This is the voltage input from an external source, between 0 and 10 volts. The digital outputs of the ADC are input to a tri-state buffer, a 74LS244, labeled IC7. We discussed how this type of buffer operated in Chapter 4, and a similar circuit was shown in Chapter 5.

The outputs of the buffer (IC7) are connected directly to the VIC-20 data bus lines, D0–D7. The buffer will become enabled when the VIC-20 electrically reads the data at the ADC outputs. The VIC-20 will read the data using the PEEK instruction, as described in Chapter 3.

In order to start the analog-to-digital conversion, the input pin 11 of IC6 in Figure 8.7 must be set to a logical 1 and then reset to a logical 0. This is the "START CONVERSION" input signal for this particular

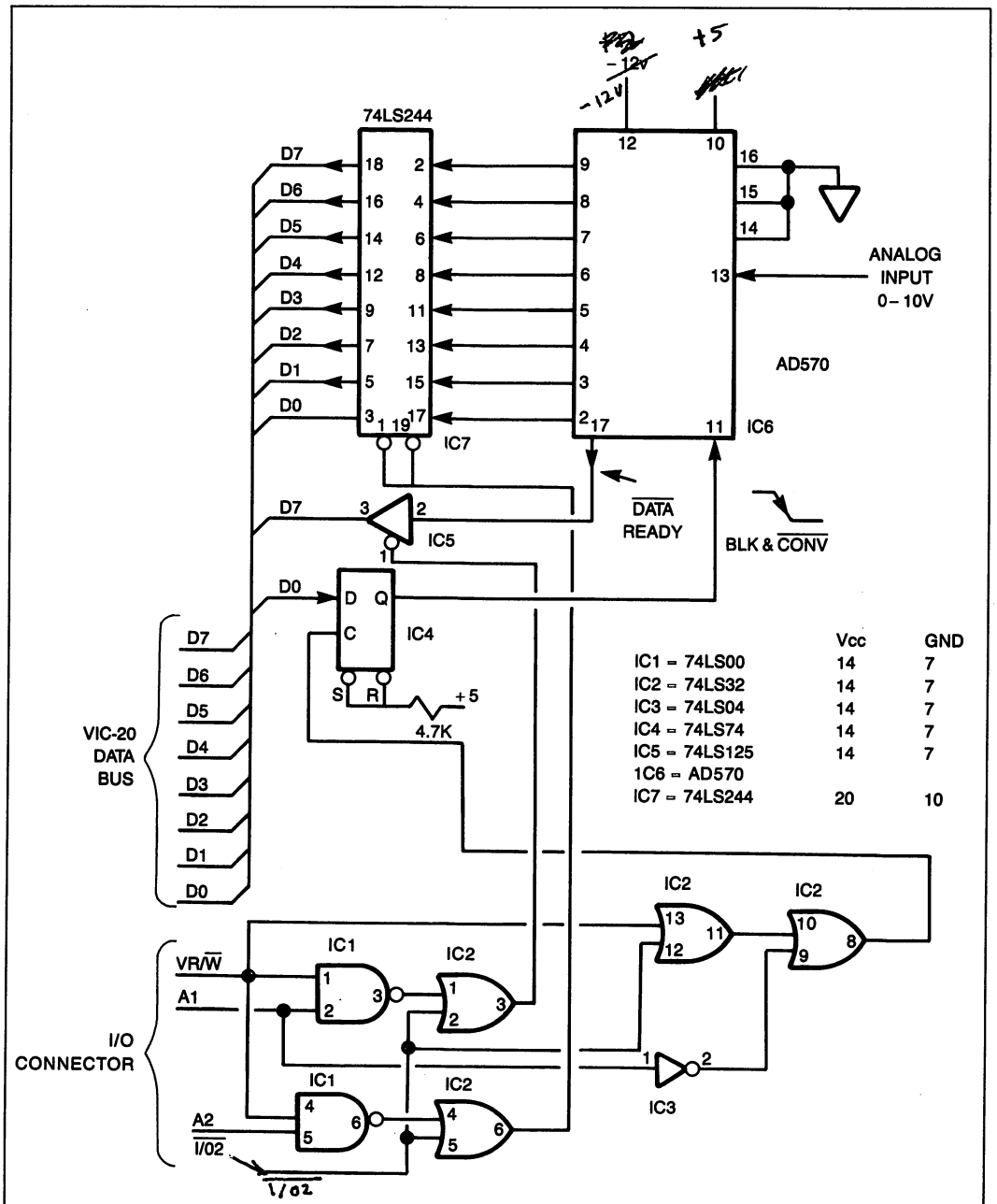



***Figure 8.7**–Complete schematic for an ADC interface to the VIC-20 computer via the expansion connector.*

ADC. When input pin 11 is reset to a logical 0, the analog-to-digital conversion will start. This is graphically shown by the wave form next to the signal line at IC6.

Input pin 11 of the ADC is connected to the Q output of a 74LS74 D flip-flop, IC4. This flip-flop acts as a single-bit latch. The data input to the flip-flop is connected to the D0 data line on the VIC-20 computer. Whenever the user wishes to set the Q output to a logical 1, the D0 line must be set to a logical 1. At the same time, the flip-flop will be clocked.

We can perform all the steps necessary to start the conversion by using the POKE instruction in BASIC. When we wish to set the Q output of the flip-flop to a logical 1, we use the weight of D0 in the POKE instruction. For example, the following instruction:

**POKE** address, 1

will set the Q output of the D flip-flop to a logical 1. Likewise, the instruction:

**POKE** address, 0

will set the Q output of the D flip-flop to a logical 0. Later in the chapter we will discuss how the address of these two instructions is computed. The important point here is that we can control an individual hardware line with a software instruction. Specifically, we can assume that the input pin 11 of IC6 can be set to a logical 1, or a logical 0, under software control.

To know when the analog-to-digital conversion is complete, the VIC-20 PC must electrically monitor the output pin 17 of IC6. This pin is the "END OF CONVERSION" signal for the ADC. Output pin 17 of IC6 in Figure 8.7 is connected to input pin 2 of IC5, a 74LS125 tri-state buffer. The data sheet for the 74LS125 is given in Appendix A. Pin 17 of the ADC will be set to a logical 0 when the analog-to-digital conversion is complete. At all other times it will be a logical 1.

The 74LS125 will perform a function similar to that of the 74LS244 mentioned earlier. That is, when the VIC-20 executes a PEEK instruction to the correct address, the 74LS125 will become enabled. This will allow the logical state of the output pin 17 of IC6 to be enabled onto the VIC-20 data bus. At that time, the VIC-20 will electrically input the logical level into a BASIC program.

We have now seen how the "END OF CONVERSION" signal is

monitored by the computer. We have also seen how the computer will set the input pin 11 of IC6 to a logical 1 or 0 under control of the software.

Each of these events will be accomplished using either a PEEK or a POKE instruction. A major part of both of these instructions is the address. We will now discuss how the address of the PEEK or POKE instruction is related to the schematic diagram given in Figure 8.7.

For the computer to read the data from the ADC, the outputs of IC7 of Figure 8.7 must be enabled. This means pins 1 and 19 of IC7 must be held at a logical 0 voltage level. If we follow the connection from pins 1 and 19 of IC7, we can see that they are connected to the output pin 6 of IC2. Output pin 6 will be a logical 0 when the $\overline{I/O2}$ line is a logical 0 and pin 6 of IC1 is a logical 0. Pin 6 of IC1 will be a logical 0 when the VR/$\overline{W}$ line is a logical 1 and the A2 address line is a logical 1. Therefore, the logical conditions that will enable IC7 pins 1 and 19 are the following:

$\overline{VR/W}$ = logical 1 (PEEK)

A2 = logical 1 (I/O2 + 4)

$\overline{I/O2}$ = logical 0

The I/O circuit has a PEEK address equal to 38912. The PEEK instruction needed to read the data from the ADC will thus be:

```
A = PEEK(38912 + 4)
```

To read the logical conditions of output pin 17 of IC6, the address line A1 (instead of address line A2) must be a logical 1. The PEEK instruction to read this line will be:

```
A = PEEK(38912 + 2)
```

The last signal we must control is the Q output of the D flip-flop, IC4 in Figure 8.7. To write a logical 1 or a logical 0 to the flip-flop, a POKE instruction is used. If we follow the clock (C) input line from the D flip-flop, we can see that it will connect to the output pin 8 of IC2. Whenever pin 8 of IC2 goes from a logical 0 to a logical 1, the information at the D input of the flip-flop will be latched at the Q output.

Address decoding of the clock line for the flip-flop is shown in the lower-right corner of Figure 8.7. Let us follow the logic of this decoding. IC2 pin 9 will become a logical 0 when A1 is set to a logical 1 via the inverter, IC3. Pin 10 of IC2 will be a logical 0 when $\overline{I/O2}$ and VR/$\overline{W}$

are both equal to a logical 0. When pins 9 and 10 are a logical 0, pin 8 of IC2 will be set to a logical 0. Pin 8 is connected to the clock input of the D flip-flop.

All of the preceding hardware conditions will be true when the POKE instruction is used as follows:

```
POKE 38912 + 2, data
```

The data will be a 1 if we wish to set the Q output to a logical 1, and a 0 if we wish to set the Q output to a logical 0.

We have discussed the hardware action taking place in the schematic shown in Figure 8.7 in some detail, to relate hardware events to software instructions. The discussion was meant for those who wished to know exactly how the hardware of the interface operates.

## 8.5 SOFTWARE FOR ANALOG-TO-DIGITAL CONVERSION

Now that we have the ADC connected to the VIC-20, let's control it with the software. We will show the BASIC software required to input a digital value from the ADC. A flowchart for the steps of this program is shown in Figure 8.8.

The following discussion will realize the flowchart of Figure 8.8 with BASIC programming statements. It is assumed that the ADC I/O device is located at address 38912 in the VIC-20.

### Step 1

The first step is to write a logical 1 to the flip-flop on the ADC I/O circuit. We can accomplish this using a POKE instruction that sets address line A1 equal to a logical 1.

```
POKE 38912 + 2,1
```

The value of 2 added to the I/O address will set address line A1 to a logical 1.

### Step 2

In this step the Q output of the flip-flop is reset to a logical 0. This

can be accomplished in the same way as in step 1, except that the data is now a 0 instead of a 1:

```
POKE 38912 + 2, 0
```

The analog-to-digital conversion has now started.

## Step 3

We must now read the logical level of pin 17 of IC6 to determine if the conversion is complete. This can be accomplished by using the following instruction:

```
R1 = PEEK (38912 + 2)
```

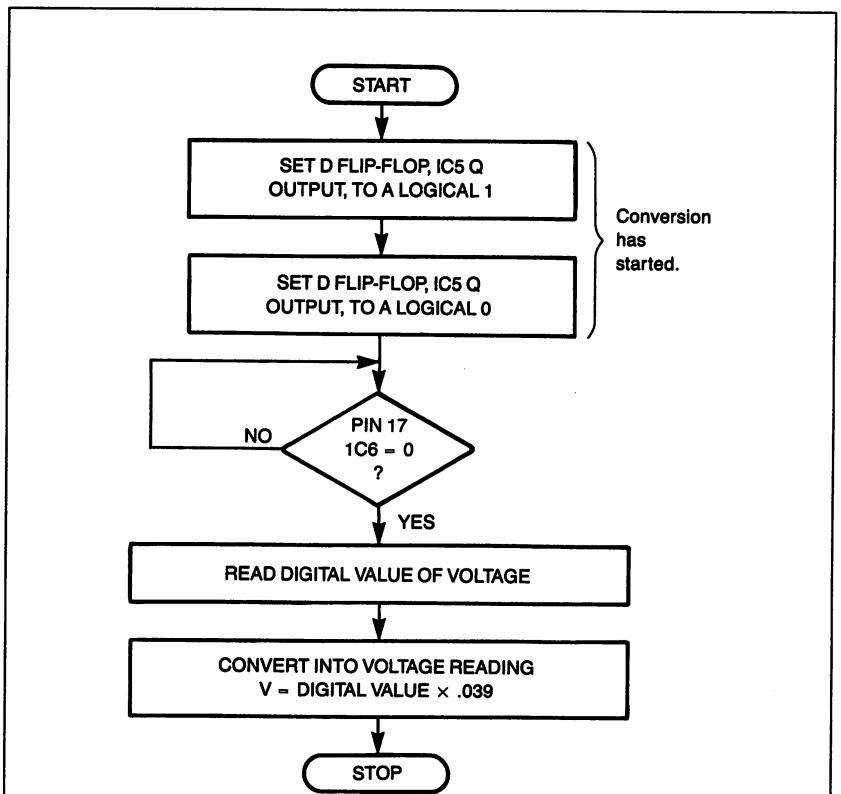



***Figure 8.8**–Flowchart showing the sequence of events required to electrically communicate with the circuit of Figure 8.7.*

Notice that when we read R1, its weight will be the summation of the data lines, D0–D7. However, as we see in Figure 8.7, we are physically controlling the logical level of only one input line during this PEEK instruction. The other input lines can be either a logical 1 or a logical 0. Further, the other input lines may not show the same value during each PEEK instruction. The actual logical level will depend on many factors, most of which are out of the user's control.

Therefore, we would like to logically ignore data lines D0–D6 during the PEEK instruction. Although we cannot electrically do this, it can be done using software. In the system software, we can ignore data lines D0–D6 by testing for a value greater than 127. (If the value of R1 is at least 128, we know that D7 must have been a logical 1.) It can be done as follows:

**IF R1 > 127 THEN GOTO (line number of PEEK instruction above)**

The value of R1 will be greater than 127 if D7 is equal to 1, and will be less than or equal to 127 if D7 equals a logical 0. Here is the reason why:

When R1 was read into the computer, by means of a PEEK instruction, it was made up of eight data input lines. Only one of these lines is of importance, D7. The input data will logically appear in one of two ways:

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| 1 | X | X | X | X | X | X | X |

or

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| 0 | X | X | X | X | X | X | X |

In these two cases the logical state of D7 was either a 1 or a 0. The other data input lines are shown as "X"; this means we do not care what their logical state is. However, their logical state is important, as it will affect the overall computed weight of the input byte.

By testing whether R1 is greater than 127 or not, we can effectively ignore all the other data lines. If those lines were all a logical 1 and D7 was a logical 0, R1 would be equal to 127. Therefore, it will not matter if lines D0–D6 are a logical 1 for this test. If R1 is a 1, then the analog-to-digital conversion is not complete and the program must loop, waiting until it is.

### Step 4

If the conversion is complete, the value of R1 will be less than or equal to 127. The digital outputs of the ADC can now be read by the computer. This will be done by the following statement:

```
V1 = PEEK(38912+4)
```

The variable V1 will be assigned the combined weight of the input data lines.

### Step 5

The final operation is to compute the analog voltage that was input. This is accomplished by multiplying the weight of V1 by the constant (10/255). We discussed this procedure earlier in the chapter; the instructions that will do this calculation are the following:

```
V2 = V1 * (10/255)
PRINT "WEIGHT = ";V1;" VOLTAGE IN VOLTS = ";V2
```

If we now put the entire set of instructions together, they will appear as shown in Figure 8.9. (It is assumed that the starting line number for this routine is 100.) The RETURN statement in line 170 was used on the assumption that all of the statements, 100–170, were contained in a BASIC subroutine. This would allow us to access the ADC at any time during a program by executing a GOSUB 100 statement.

```
100  POKE 38912+2, 1
110  POKE 38912+2, 0
120  R1 = PEEK(38912+2)
130  IF R1 = 1 THEN 120
140  V1 = PEEK(38912+4)
150  V2 = V1 * (10/255)
160  PRINT "WEIGHT = ";V1;" VOLTAGE = ";V2
170  RETURN
```

***Figure 8.9**–BASIC program for an analog-to-digital conversion. This routine implements the flowchart shown in Figure 8.8.*

## 8.6 A TEMPERATURE-MEASURING CIRCUIT (TRANSDUCER)

In this section we will discuss a circuit that can be used to sense temperature. The circuit will produce an analog output voltage that is a function of the temperature. To put it another way, the voltage produced by the temperature-sensing device will be mathematically related to the temperature measured.

The circuit we will use is shown in Figure 8.10 parts (a) and (b). Part (a) shows the device pinout, and part (b) shows how to connect the devices as a temperature-sensing transducer. The device is a National Semiconductor LM135, LM235, or LM335. Each device will work; the difference is in the temperature range (and the price) of each one.

The specified temperature ranges are:

LM135H $-55$ to $+150°$ C

LM235H $-40$ to $+125°$ C

LM335H $-10$ to $+100°$ C

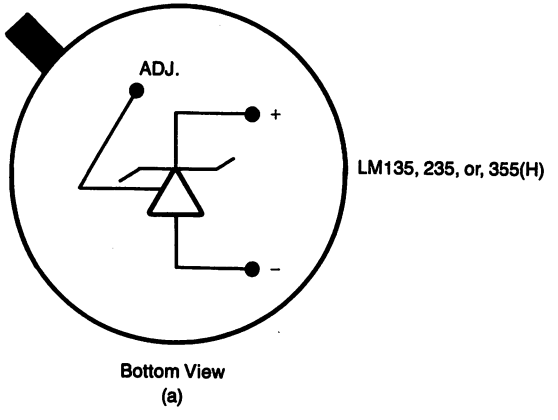
The "H" denotes that the device is packaged in metal. A model number ending in "Z" would indicate that the device was packaged in ceramic. Either of these packages will work equally well for this application.

All of the circuit components shown in Figure 8.10(b), such as the resistors R1 and R2, are located at distance from the transducer, as shown in Figure 8.11. The reason for taking this precaution is simply that resistors may change their value as a function of heat, which could affect the accuracy of the temperature sensor's reading if the two devices are placed too close together.

The voltage out ($V_{OUT}$ in Figure 8.10(b)) is connected to the ADC circuit we discussed in Section 8.4. The mathematical relationship between $V_{OUT}$ and the temperature to be measured is quite simple:

$$\frac{\text{T (in degrees Kelvin)}}{100} = V_{OUT}$$

where degrees Kelvin = 273.2 + degrees Centigrade. Using this equation, we can calculate the voltages of some known temperatures, as shown in Figure 8.12. This table is presented to show you the relationship that exists between the different units that we must work



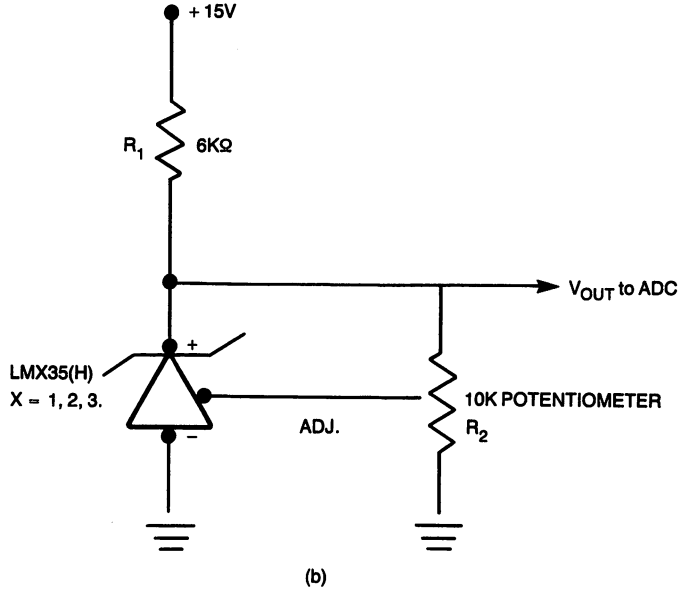




***Figure 8.10**–a) Pinout of the LM335.*
*b) Schematic diagram showing how the LM335 can be connected to act as a temperature transducer.*

with in this application. When the temperature is measured, converted, and input to the VIC-20 computer, we will know the digital weight. From this, the output voltage of the transducer can be calculated as shown in the previous section of this chapter.

Once the voltage is calculated, the temperature can be calculated in degrees Centigrade using the following set of equations:

a. $\frac{273.2 + \text{degrees Centigrade}}{100} = V_{OUT}$

b. $100 * V_{OUT} = 273.2 + \text{degrees Centigrade}$

c. $(100 * V_{OUT}) - 273.2 = \text{degrees Centigrade}$

d. $V_{OUT} = (\text{digital word}) * (10/255)$

e. $\text{Degrees Centigrade} = (100 * (\text{digital word}) * 10/255) - 273.3$

This type of mathematical calculation is very simple to perform using BASIC on the VIC-20. In fact, as we will see, only the last equation and a PRINT statement must be added to our subroutine to display the temperature.

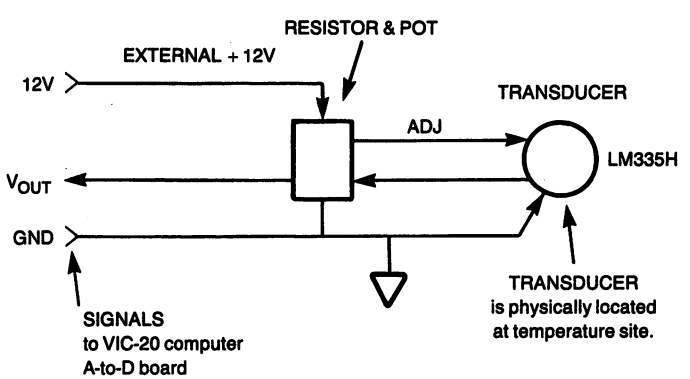



***Figure 8.11**–Block diagram showing how the circuit components of Figure 8.10(b) are located at a distance away from the actual temperature measuring site. This keeps the temperature being monitored from affecting the external components of the circuit.*

## 8.7 THE COMPLETE SYSTEM FOR TEMPERATURE MEASUREMENT

We are now ready to connect the entire system together to perform the function of automatic temperature measurement. When the system is connected via the hardware, the software will enable you to measure and record any temperature in degrees Centigrade. Some examples of temperature monitoring are given at the end of this chapter.

Figure 8.13 shows the block diagram of the complete hardware system. We have already discussed each major block of Figure 8.13. Using this block diagram, let us write a complete BASIC program that will accept the data from the ADC, and calculate and finally print the temperature in degrees Centigrade. This program is presented in Figure 8.14.

It should be pointed out that no attempt has been made to minimize the number of BASIC statements in the program. The program is intended simply to instruct. In that light, it is written in a very straightforward manner, and documented at each step with REM lines. The subroutine at line 200 may be called whenever a temperature is needed.

| TEMPERATURE | | |
|---|---|---|
| DEGREES CENTIGRADE | DEGREES KELVIN | VOLTAGE $V_{OUT}$ |
| 0 | 273.2 | 2.73 |
| 25 | 298.2 | 2.98 |
| 50 | 323.2 | 3.23 |
| 75 | 348.2 | 3.48 |
| 100 | 373.2 | 3.73 |

***Figure 8.12**–This table illustrates the relationship between the temperature measured, and the voltage output, by the temperature transducer.*

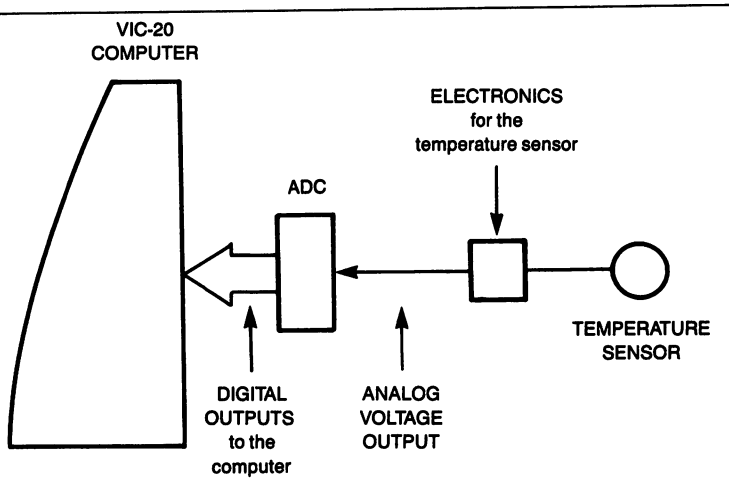



***Figure 8.13*** *–Block diagram of the complete system required to automatically measure temperature with the VIC-20 computer.*

```
 10 GOSUB 200
 20 PRINT "TEMPERATURE IN DEGREES CENTIGRADE = ";T1
 30 STOP
 40 REM THE SUBROUTINE FOR TEMPERATURE MEASURING
200 POKE 38912+2,1
210 POKE 38912+2,0
215 REM THE ABOVE WILL START THE ADC CONVERSION
220 R1 = PEEK (38912+2)
230 R1=R1 + 1
240 IF R1=1 THEN 220
245 REM THE ABOVE WILL WAIT UNTIL CONVERSION IS COMPLETE
250 V1=PEEK(38912+4)
255 REM THE ABOVE WILL READ THE DIGITAL WEIGHT
260 V2=V1 * (10/255)
265 REM THE ABOVE WILL CALCUALTE THE VOLTAGE INPUT
270 T1= (100 * V2) – 273.2
280 RETURN
290 REM THE ABOVE CALCULATES THE TEMP IN DEGREES CENTIGRADE
```

***Figure 8.14–****This program will accept an input voltage from an ADC, and calculate and finally print the corresponding temperature in degrees Centigrade.*

## 8.8 SOME PRACTICAL ADC APPLICATIONS

One application for monitoring temperature in your home is to chart a temperature profile of the house. This will allow you to better determine where problem areas of heat escape are located. To accomplish this you can connect one or more temperature probes to the VIC-20. Each probe can be placed in a certain area of the home. When the probes are in place, you can program the computer to monitor the temperature for a 24-hour period.

The temperature could be sampled at 10-minute intervals, with the computer storing the resulting values on a disk or cassette. At the end of the 24-hour period, the computer could be programmed to plot the temperature against the time of day (or any other meaningful axis). When all of the important rooms and areas in the home have been profiled, you could then make whatever physical adjustments are necessary to maintain the desired temperature.

A second application for a temperature probe would be to monitor the temperature outside the home vs. the temperature inside the home. If the desired inside temperature were cooler than the outside temperature, the computer could be made to let the warmer outside air heat up the home instead of the furnace.

Besides the temperature probe, there are many home computer applications for using analog-to-digital conversion. Some of these applications are listed below:

1. Sound detection. Sound could be converted into an analog voltage by a microphone (another kind of transducer). If the sound reached a certain level, the computer could take some action. This could be part of a security system that detected the noise of an intruder.

2. Wind direction. You could make a transducer that would produce an analog voltage directly proportional to the direction of the wind. For example, 5.0 volts = north, 3.75 volts = east, 2.5 volts = south and 1.25 volts = west. All other compass directions would be scaled accordingly.

3. A barometer. You could connect a transducer that would produce an analog voltage proportional to the barometric pressure.

4. Moisture measurement. Using the correct transducer, the moisture of the soil could produce an equivalent analog voltage. This would allow the computer to determine when the sprinkler system should be turned on.

These are just a few of the many applications for temperature measurement and analog-to-digital conversion that can be accomplished with the home computer.

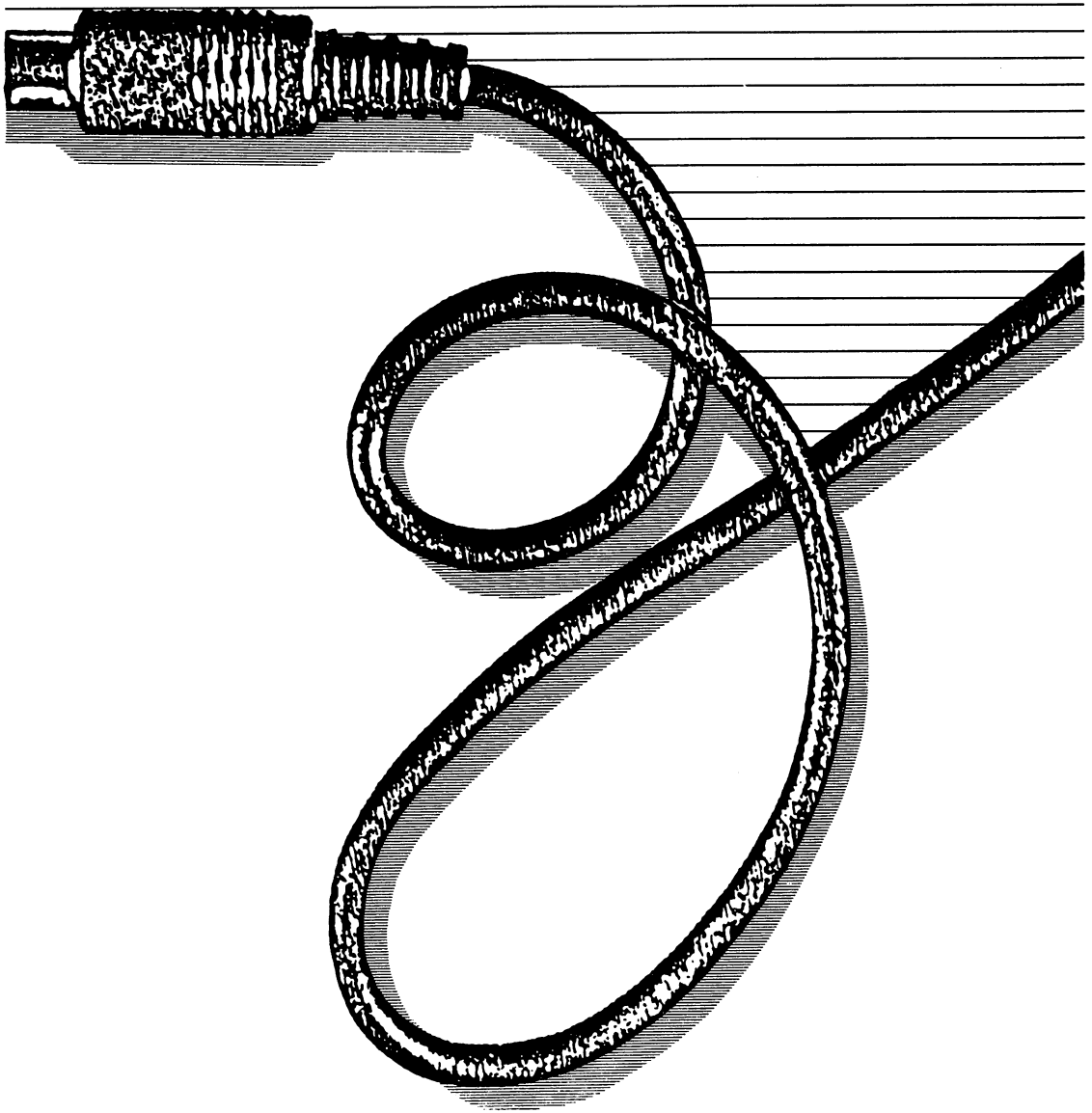
## 8.9 SUMMARY

In this chapter we have discussed the details of analog-to-digital conversion. The discussion started by explaining the need for analog-to-digital conversion. Next we discussed conceptually how it is accomplished. This discussion was at the user's level; we did not attempt to explain the details of how an ADC operates internally.

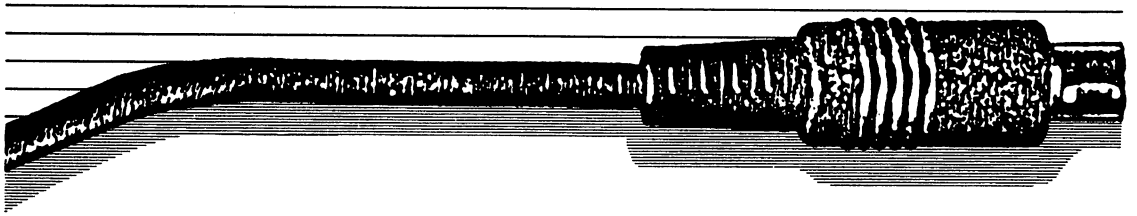
Next we proceeded to connect a real ADC, the Analog Devices AD570, to the VIC-20 Personal Computer. Actual circuits were designed and discussed. It was shown how to calculate the input voltage based on the digital word or weight read from the ADC. A temperature transducer was then connected to the ADC. A complete temperature-monitoring system was presented, along with the software for controlling the system. Finally, we suggested a few practical applications for a system involving an ADC and a VIC-20.

The main focus of this chapter was to show how to perform analog-to-digital conversion using a VIC-20 computer. The example of inputting a temperature was meant only as a single illustration of how ADC may be employed in computer control. Whatever they measure, all transducers generate electricity, and whenever an analog voltage needs to be monitored by a digital computer, it can be done in a manner similar to that shown in this chapter.

# DIGITAL-TO-ANALOG CONVERSION FOR THE VIC-20 COMPUTER



# 9



Certain peripheral devices that can be controlled by a home computer require an analog input voltage in order to operate. (We discussed what is meant by an analog voltage in Chapter 7.) A home computer system is completely digital, which presents us with an interfacing problem similar to the one we discussed in the last chapter. What we need now is a means of producing an analog voltage from a digital source or controller. Such a process is called *digital-to-analog conversion,* or more simply, *DAC*. Figure 9.1 shows a block diagram of this concept. Digital-to-analog conversion is the mirror image of analog-to-digital. They are similar but exactly opposite concepts.

One type of external device that might require an analog input voltage would be a direct current (DC) motor. Most DC motors have the operating characteristic of rotating faster when a higher voltage is applied to the input. A simple schematic of a DC motor, along with its analog input, is shown in the block diagram of Figure 9.2.

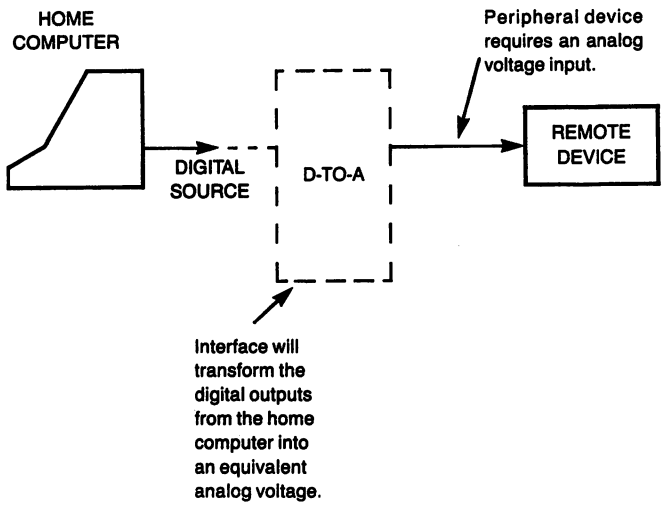



***Figure 9.1**–Block diagram showing the interfacing problem that occurs when a completely digital source needs to control a peripheral device that requires an analog voltage input. A DAC will solve this interfacing problem.*

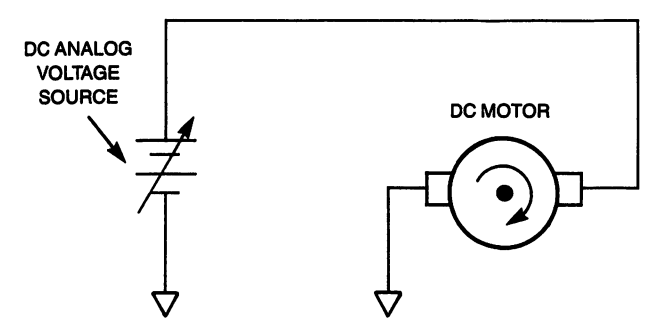



***Figure 9.2**–Diagram showing a DC motor being controlled by an analog voltage. When the voltage increases, the motor will increase in speed. This is only one type of peripheral device that can be controlled by an analog input voltage.*

We have shown only a single example of an external device that would require an analog voltage, but there are many other applications using devices that require an analog voltage for control. For instance, communications and recording systems utilize digital-to-analog conversion and analog-to-digital conversion at their input and output. Electronic music and waveform-generation systems also use digital-to-analog conversion as a part of the overall system. These systems are growing in popularity. In short, any device whose output (speed, musical pitch, brightness, etc.) is variable will need digital-to-analog conversion to be controlled by digital electronics.

The intention of this chapter is to show how you can control and set an analog output voltage from a digital source, such as the VIC-20 computer. Our discussions will cover the basics of digital-to-analog conversion. We will design and discuss the hardware and software necessary to allow the VIC-20 to automatically control an analog output voltage from an output connector. The circuits shown are inexpensive and available from many suppliers. These circuits will show one way to achieve digital-to-analog conversion. Nothing in this chapter will be left up to the reader; no important detail will be omitted. All pin numbers, connecting lines and types of devices will be labeled.

## 9.1 WHAT IS DIGITAL-TO-ANALOG CONVERSION?

Before we design and discuss a digital-to-analog conversion interface in detail, let us examine the overall concept. In the introduction to this chapter a very informal definition was given. This section will bring that definition into sharper focus.

Figure 9.3 shows a general block diagram of a digital-to-analog converter. In this diagram the block labeled "B1" is the electrical interface to the digital source that will input data to, and control, the converter. In our case the source will be the digital output lines from the VIC-20 computer. Block B1 will control the block labeled "B2" in Figure 9.3.

Block B2 is the precision scaling network. The job of the scaling network is to electrically determine how much of the reference voltage or current will appear at the converter output. For example, suppose the reference voltage is equal to 8.00 volts. The scaling network

is capable of setting the output voltage to an exact portion of the reference, say 2.04 volts. The reference could also be a smaller voltage than the output device requires, such as 1.5 volts. In this case, the scaling network would select a portion of the 1.5 volts, and this would be amplified before being output from the DAC.

Digital inputs from the computer will electrically control the scaling network, determining how much of the reference will be applied to the output. By setting the proper digital information on the DAC inputs, we could force the DAC output voltage to be equal to exactly half of the 8.00-volt reference, or 4.00 volts. We cannot give a formal, general definition of what voltage will appear at the output as a function of the digital inputs, because different digital-to-analog converters may have different specifications.

Using what we know about digital-to-analog conversion, let us take

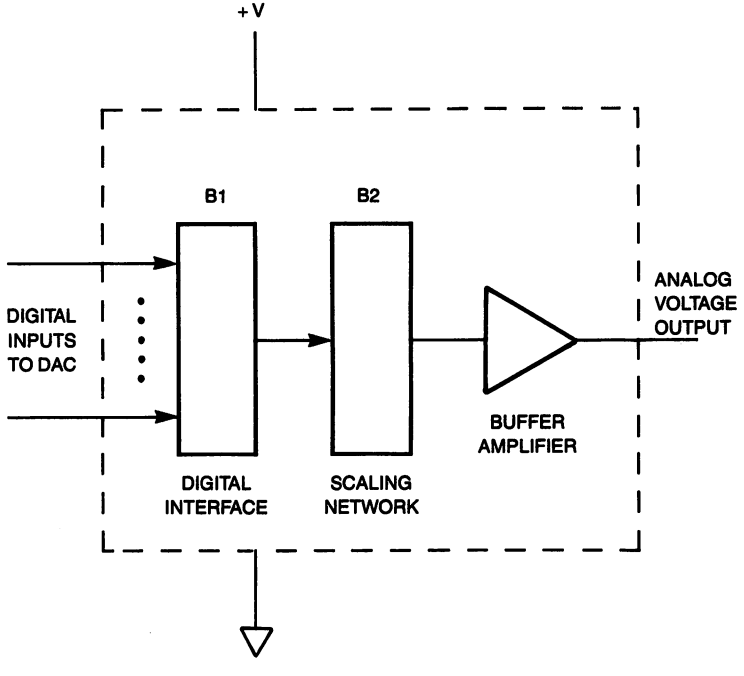



***Figure 9.3**–Block diagram of a general digital-to-analog converter. Most DACs will have a block diagram similar to this one.*

an example and show how to set the analog output voltage as a function of the digital inputs. Suppose we have a digital-to-analog converter that will output 10.00 volts when all of its digital inputs are set to a logical 1; that is, suppose we have a DAC whose reference voltage is 10.00 volts. (We use this as an illustration only. Every DAC has a unique set of specifications.) Let us further assume that the DAC will output 0.0 volts when all of the digital inputs are a logical 0. Figure 9.4 shows these two conditions. A DAC with this type of voltage

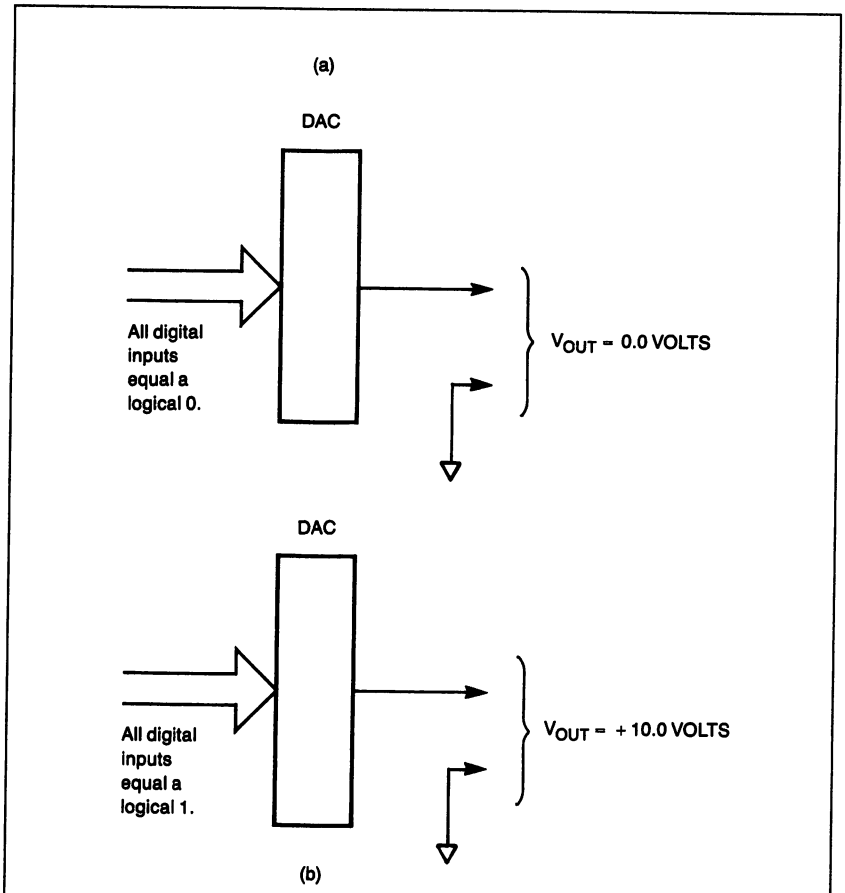



***Figure 9.4**—a) When all digital inputs to the DAC are a logical 0, the output voltage will equal 0.00 volts.*
*b) When all digital inputs to the DAC are a logical 1, the output voltage will equal 10.00 volts. These are the extreme ranges of this DAC's output voltage swing.*

output specification is called unipolar. The output voltage will swing in one direction (toward one electrical pole) only. In this case the swing is from 0.0 volts to +10.0 volts. The term "unipolar" also describes an analog-to-digital converter with the same voltage characteristics.

Next we need to know how many digital inputs the DAC has. We will assume that this DAC has 10 digital inputs. There are therefore $2^{10}$, or 1024, different input combinations that can be used. (See Figure 9.5.) From this number we can calculate the minimum voltage swing, the increment by which the voltage will change for each unique digital combination on the input.

To calculate how much the DAC output voltage will change when the digital input combination changes by only one least significant bit, we can use the following formula:

1. Minimum output voltage change = maximum output voltage change/maximum number of states.
2. Maximum output voltage change = $V_{OUT}$ max − $V_{OUT}$ min
3. $V_{OUT}$ min = 0.0 volts , $V_{OUT}$ max = 10.0 volts

   Maximum output voltage change = 10.0 − 0.0 = 10.0

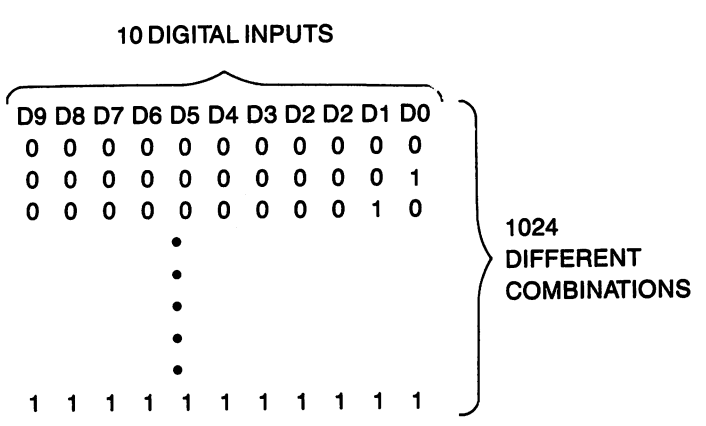



***Figure 9.5**–With 10 digital inputs to the DAC, there are 1024 different combinations that may be applied. The number 1024 is derived from raising 2 to the tenth power.*

4. Maximum number of unique digital combinations with 10 inputs = $2^{10}$, or 1024. One of these combinations is used to output 0.0 volts. Therefore, we have actually 1024 – 1, or 1023 digital values that produce a voltage greater than 0.
5. Minimum voltage swing = 10.0 volts/1023 = .009775 volts, or 10 millivolts.

This means the output voltage will change by 10 millivolts for each digital input bit that changes. (See Figure 9.6.) This DAC would be described as a unipolar, 10-bit, voltage DAC. (The term "voltage" indicates that the output is a voltage and not a current.) Suppose our hypothetical DAC had 12 inputs instead of 10. It would then have $2^{12}$, or 4096, possible digital input combinations. Using the formulas given, the minimum voltage change at the output would be equal to:

6. Minimum voltage change = 10.0 volts/(4096 – 1) = 10/4095 = .0024 V

Let us now suppose that this same DAC had only 6 inputs. This would allow $2^6$, or 64, different digital input combinations. The minimum voltage change at the output pin would be:

7. Minimum voltage change = 10.0 volts/(64 – 1) = 10/63 = .158 V

Notice that, for a given output voltage range, the greater the number of digital inputs, the smaller the minimum voltage change at the DAC output. (See Figure 9.7.) In general, the greater the number of digital input lines, the better the output voltage "resolution"; that is, the DAC can resolve a smaller increment of voltage. This means we can come closer to obtaining the "smooth" staircase waveform shown at the top of Figure 9.7.

## 9.2 AN ACTUAL DIGITAL-TO-ANALOG CONVERTER

To illustrate the points we have covered about digital-to-analog conversion, let us examine a real DAC that is available "off the shelf." The DAC chosen for this example is the AD558, manufactured by Analog Devices. A complete data sheet for this device is given in

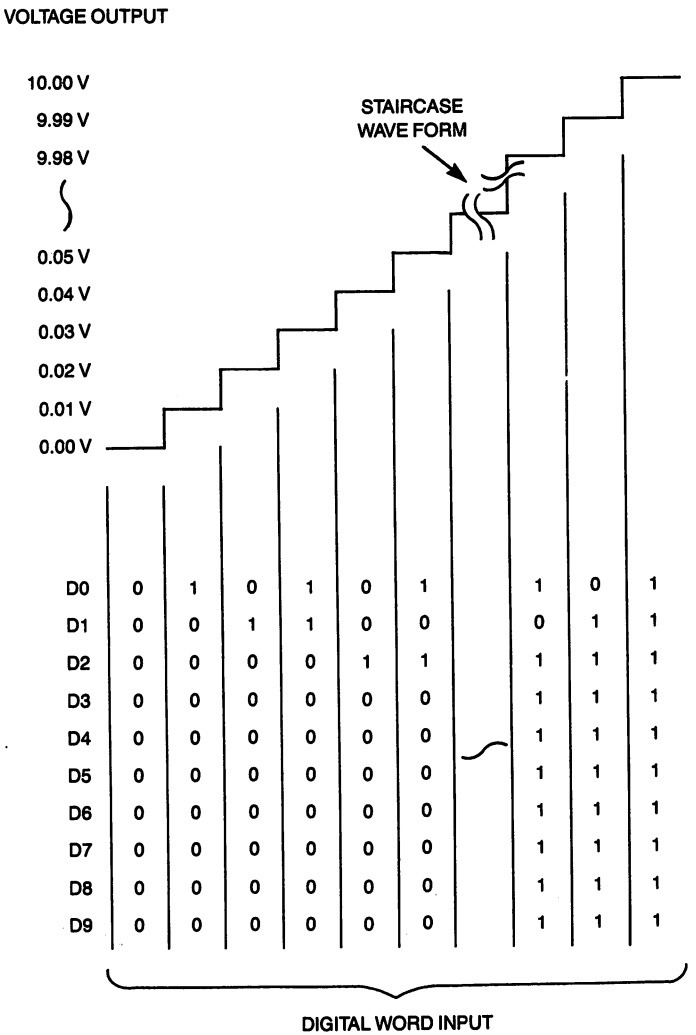



***Figure 9.6**–The relationship between the digital input values and the analog output voltage for a DAC with 10 inputs and an output voltage swing between 0.00 and + 10.00 volts.*

Appendix A of this text. A block diagram of the AD558 is shown in Figure 9.8; let us discuss it in detail.

In this figure we see the digital input block, labeled DB0 – DB7. These digital inputs will connect to the data lines of the VIC-20. This DAC has built-in latches that allow the data output from the VIC-20 to be strobed and electrically stored without the use of an external latch. (See Figure 9.9.) This is a very helpful feature of the AD558, because it reduces the total number of parts necessary. We will show exactly how to connect the data lines of the VIC-20 to the DAC digital inputs in a later section of this chapter.

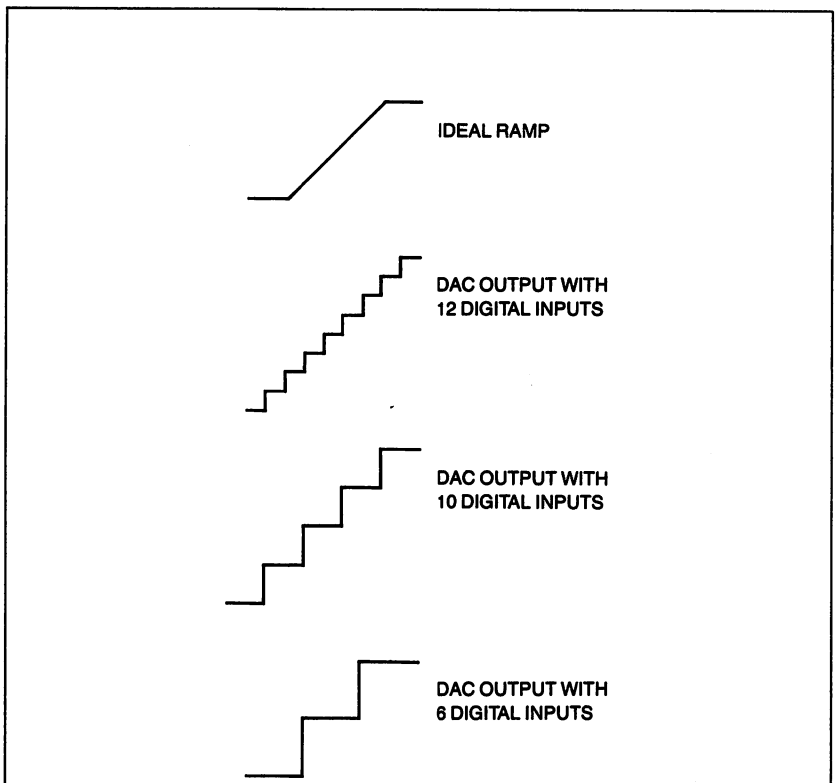



**Figure 9.7**–*Different "staircase" wave forms for the analog voltage output of the DAC. As the digital input word changes by a single count, the output will increase by the minimum voltage change. Notice that as the number of digital inputs to the DAC increases, the staircase more closely resembles the ideal ramp.*

The next block shown in Figure 9.8 is the *control logic.* This logic block will provide the strobe signal to the input storage latches at the correct time. The VIC-20 will output a strobe pulse to the control logic block.

Another important block of Figure 9.8 is the *band-gap reference.* This is the internal reference voltage for the DAC. With the reference voltage inside the package, we do not have to provide an external precision reference for the DAC. This feature is useful because it means that standard power supplies can be used to power the device. Most digital electronic systems have power supplies to suit this application. The VIC-20, however, does not have a + 12 volt supply available at its output connector. Therefore, an external + 12 volt supply must be used.

A large block of Figure 9.8 is the *8-bit voltage switching and D-to-A*

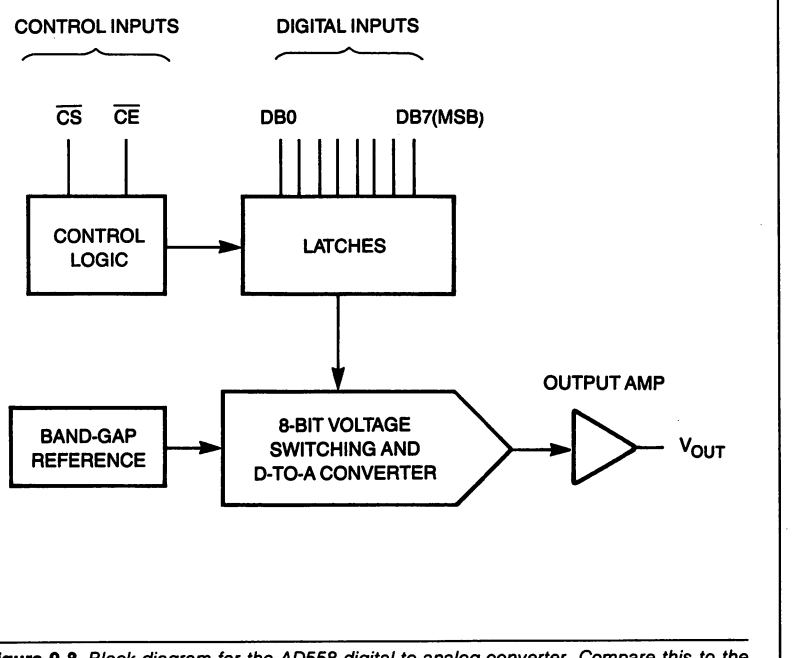



***Figure 9.8**–Block diagram for the AD558 digital-to-analog converter. Compare this to the general block diagram given in Figure 9.2.*

*converter*. This block is the scaling network that applies a portion of the reference voltage to the output. In this device the voltage is output to an external device (not shown) from an amplifier, labeled "output amp" in Figure 9.8. At this point in the discussion, you should compare the block diagram of Figure 9.8 to the general block diagram that was given in Figure 9.2.

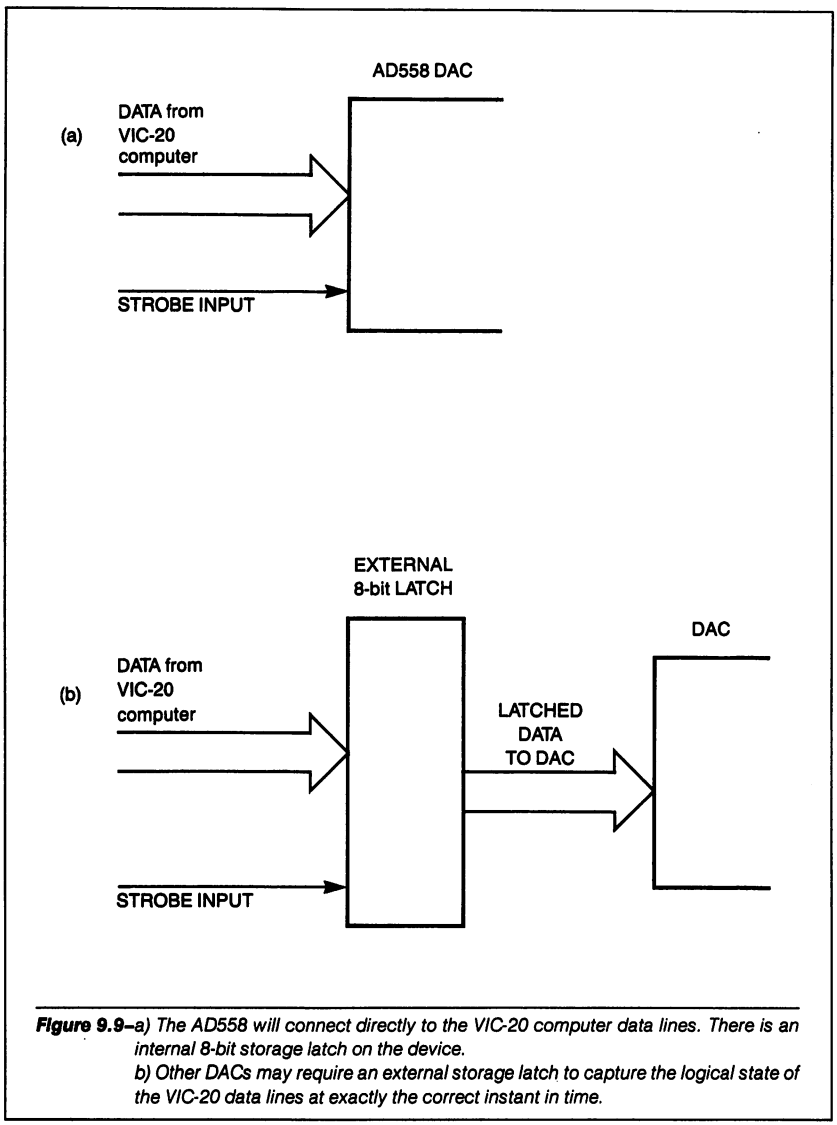



***Figure 9.9**–a) The AD558 will connect directly to the VIC-20 computer data lines. There is an internal 8-bit storage latch on the device.*
*b) Other DACs may require an external storage latch to capture the logical state of the VIC-20 data lines at exactly the correct instant in time.*

There are two modes of operation for the AD558. One mode allows the output voltage to swing between 0.00 volts and +2.56 volts. The second mode allows the output voltage to swing between 0.00 volts and +10.24 volts. Figure 9.10 shows how to connect the pins of the AD558 for an output voltage swing in each of the two

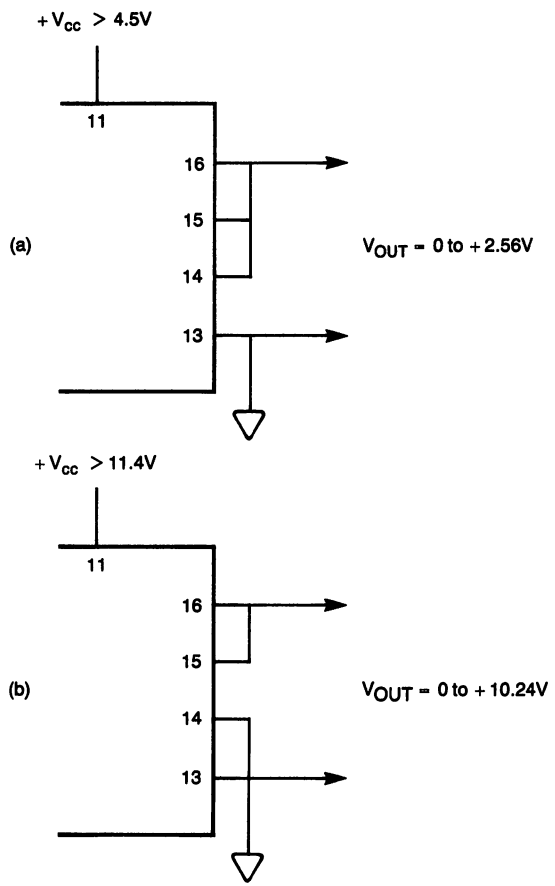



***Figure 9.10**–a) Wiring diagram for using the AD558 with a maximum output voltage of 2.56 volts.*
*b) Wiring diagram for using the AD558 with a maximum output voltage of 10.24 volts.*

modes. The 10.24-volt mode requires the $V_{cc}$ input voltage pin to have a potential between 11.4 and 16.5 volts. The 2.56-volt mode requires a minimum potential of 4.6 volts.

## 9.3 CONNECTING THE DAC TO THE VIC-20

Let us now discuss how to connect the AD558 to the VIC-20 PC for automatic control. This discussion will describe how to physically wire the AD558 to a VIC-20 output expansion connector. The information presented can be applied to connecting the VIC-20 computer to most other DACs as well. Figure 9.11 shows the complete schematic diagram for connecting the AD558 to a VIC-20.

We see in Figure 9.11 that the data lines from the computer, D0– D7, are connected directly to the data input pins of the device. We can do this because of the AD558's built-in 8-bit latch, which will store the logical conditions of the data lines at the correct time. Other DACs may not have this feature. Digital-to-analog converters without built-in storage latches require an external 8-bit latch, used as shown in Figure 9.12.

To strobe the data into the DAC circuit shown in Figure 9.11 or 9.12, the VIC-20 will execute a POKE instruction. The use of the POKE instruction was described in Chapter 2. Timing considerations for latching the data during a POKE instruction were discussed in Chapter 4. When the POKE instruction is executed, the data specified by the instruction will be placed at the AD558 data inputs via the eight VIC-20 data lines.

At this time the $\overline{I/O2}$ line, as well as the VR/$\overline{W}$ control line, will go to a logical 0. When this occurs, the CS input pin 10 of the AD558 will be set to a logical 0. After the POKE instruction is complete, the $\overline{CS}$ input line to the AD558 will be set to a logical 1, because the VR/$\overline{W}$ control line will go to a logical 1 as well as the $\overline{I/O2}$ line. The operation of the $\overline{I/O2}$ line and the VR/$\overline{W}$ control line was discussed in Chapter 4.

When the CS input line to the AD558 goes to a logical 1, the data at the input pins will be strobed into the internal 8-bit latch. The output voltage at pin 16 will depend on the data written to the DAC during the execution of the POKE instruction. By using the POKE instruction and the circuit shown in Figure 9.11, we can set any digital input combination between 0 and 255 at the DAC input pins. Now we need to

know how to calculate exactly what data should be written to set any voltage at the DAC output.

## 9.4 SETTING ANY OUTPUT VOLTAGE ON THE DAC

We will assume that the VIC-20 computer system can control the digital inputs to the DAC as discussed in Section 9.3. The DAC will be

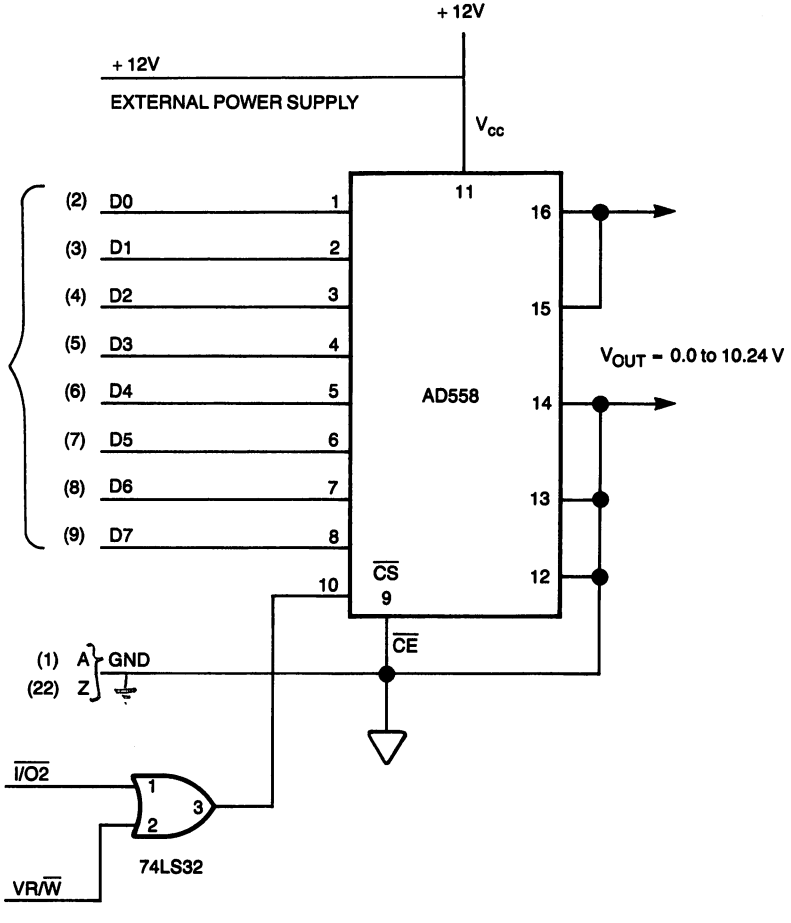



***Figure 9.11**–Complete schematic for interfacing the AD558 to the VIC-20 computer via the expansion connector.*

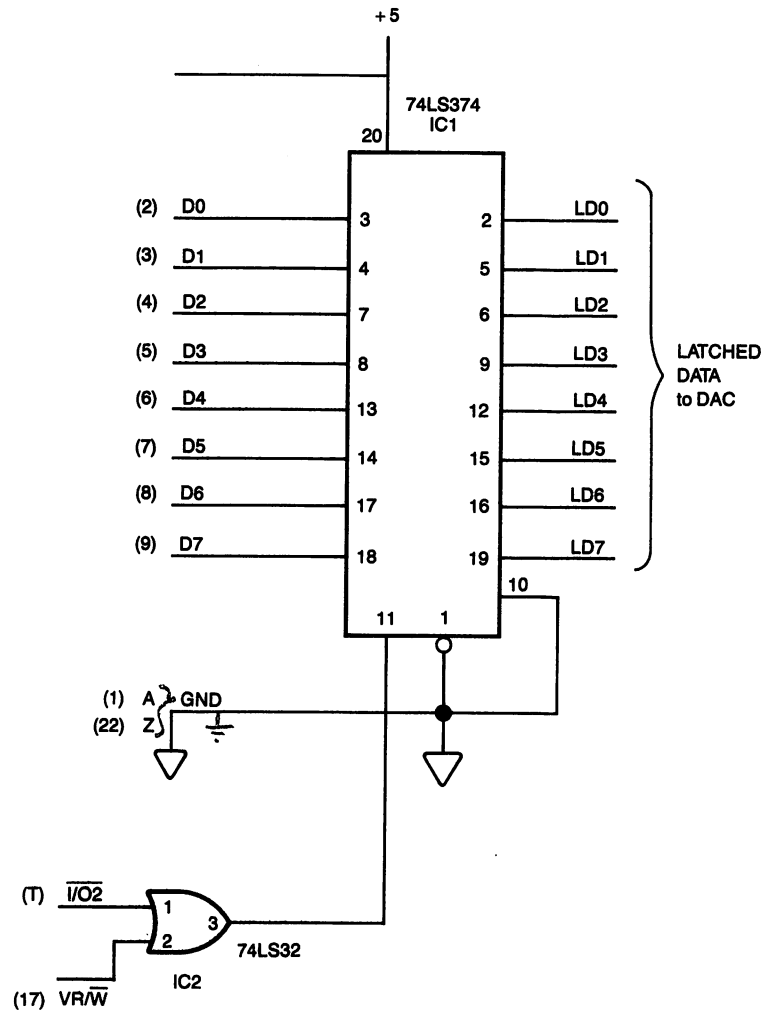



***Figure 9.12**–Schematic diagram showing how an external latch could be connected to the VIC-20 computer for use with a DAC that, unlike the AD558, does not have an internal storage latch.*

physically connected to the system as shown in Figure 9.11. If the VIC-20 can control the digital inputs to the DAC, how can we calculate the digital byte needed to set the correct voltage output? That is the question we will answer in this section.

In Section 9.1 we discussed briefly how to calculate the minimum voltage change that will occur at the output of the DAC. That discussion assumed there were 10 digital inputs to the device, but the AD558 has only 8 inputs. Therefore, the equations given in Section 9.1 will need to be modified slightly to handle the differences in the two examples. We could interface a 10-bit DAC to the VIC-20 computer, but it would require additional hardware, because the VIC-20 data bus handles only 8 bits at a time. DACs or ADCs handling any number of bits can be used with the VIC-20, but we have used 8-bit DACs and ADCs here to illustrate the concepts, because these devices interface directly to an 8-bit data bus.

Our objective in this section is to calculate the total weight of the digital inputs required to output a certain voltage from the DAC. For example, suppose we wish the DAC's output voltage to be equal to 4.8 volts. The computer will be programmed to output a certain combination of logical 1s and 0s to obtain the correct DAC output voltage. The digital output byte in this case would be equal to 120. Let us go through the steps needed to perform this calculation. We will be using the AD558 DAC as the example. However, you can easily modify the equations given to fit any DAC with a different number of digital inputs and a different output voltage swing.

The first step is to calculate the minimum voltage change at the DAC output. This can be accomplished using the procedure outlined in Section 9.1, and the appropriate specifications for the AD558.

$$\text{Minimum voltage swing} = \frac{\text{Maximum voltage swing}}{\text{Total number of digital input combinations} - 1}$$

The maximum voltage swing on the AD558 will be equal to 10.24 volts. There are eight digital inputs to the AD558. (That is why it is called an *8-bit* DAC.) With eight digital inputs there are a total of 256 different input combinations. One of these combinations is used to set the DAC to its lowest value, so we use 255 (256 – 1) combinations to generate the output voltage swing.

Minimum voltage swing = 10.24/255 = .040 volts (40 millivolts)

This means that if the digital input byte to the DAC were 00000001, the output voltage would be 40 millivolts.

Given the digital input word, we can calculate the DAC output voltage by the equation:

$$V_{OUT} = (\text{Digital input word}) \times .040 \text{ volts}$$

For example, let us suppose we have a digital input word equal to 65. The output voltage is calculated as $65 \times .040 = 2.60$ volts.

However, we are interested in the converse of the preceding case: Given a voltage needed, what digital input word is required by the DAC? To calculate this, we use the relationship between the voltage desired and the known voltage output change for each digital input change. The analog output voltage for the DAC will change .040 volts for each digital count. Therefore, we can divide the voltage needed or desired at the DAC output by .040. The result will be the digital value needed to produce the output voltage.

For example, suppose we wished the DAC output voltage to be equal to 8.00 volts. To find out what digital word is needed by the DAC, we set up the relationship:

$$\text{Digital word} = \frac{V_{OUT} \text{ wanted}}{.040}$$

In our example the equation would be:

$$\text{Digital word} = \frac{8.00}{.040} = 200$$

If we input the digital weight of 200 to the DAC data input lines, the DAC output voltage will be equal to 8.00 volts. This type of calculation is quite easy to do with the VIC-20. We will present a program in the next section that will perform this calculation.

Let us suppose that we wished the DAC to output a voltage equal to 5.25 volts. To calculate the digital word we would need to use the equation:

$$\text{Digital word} = \frac{5.25}{.040} = 131.25$$

Notice that the digital word is not an integer, but a real number. That

is, the number has some digits that are not zero after the decimal point. To control the DAC with the VIC-20 computer, we must output digital words as integers: 1, 2, 234, 179, etc. The exact voltage we require at the output of the DAC cannot be obtained. We must choose whether we wish the output voltage to be a little larger than 5.25 or a little less than 5.25. (See Figure 9.13.)

The integer numbers that we must choose between are 131 and 132. Neither of these two numbers will give the exact output voltage of 5.25 volts. The number 131 will give an output voltage equal to 131 $\times$ .040 = 5.24 volts. The number 132 will give an output voltage equal to 132 $\times$ .040 = 5.28 volts. It is clear that the integer 131 gives an output voltage value closer to 5.25 volts than does 132.

As a general rule, we can round off to the closest whole number and use the rounded-off value as the digital word to be output. If the decimal part of the required digital word is less than .5, use the smaller integer. If the decimal part of the digital word is greater than or equal to .5, use the greater integer as the output word.

To illustrate, let us suppose the required digital word was equal to 156.34. The decimal part of this word is equal to .34, so we will use the smaller integer, 156, as the output word. If the required digital word was equal to 156.67, we would use the greater integer, 157, as the output word. This process, testing the decimal value and output-

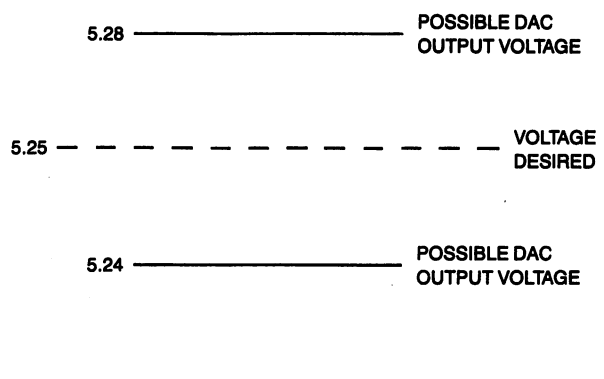



***Figure 9.13**—The desired output voltage is 5.25 volts. The DAC will output a voltage of either 5.24 volts or 5.28 volts. We must choose which of these two output voltages is the most desirable for the application.*

ting the nearest integer, can be accomplished quite easily with the VIC-20, using BASIC's INT function. The program given in the next section, which calculates the digital value needed, uses this function. The same type of calculations for the digital input word and the rounding-off of the word to integer values may be applied to a DAC with any number of input lines and any output voltage swing.

## 9.5 CONTROLLING THE DAC WITH A BASIC PROGRAM

In this section we will present a BASIC program that will allow the AD558 to be programmed to any output voltage between 0.00 and 10.24 volts. The program will perform the calculations described in Section 9.4, using the information presented there. The program is shown in Figure 9.14.

```
 10 REM THIS PROGRAM WILL INPUT A NUMBER FROM THE KEYBOARD
 20 REM AND POKE THE FORMATTED DATA TO THE I/O SLOT.
 30 REM
 40 REM
 50 REM
 60 PRINT"INPUT THE VOLTAGE FOR THE DAC"
 70 PRINT"IT MUST BE BETWEEN 0 AND 10.24 VOLTS"
 80 PRINT
 90 INPUT V1
100 REM
110 REM NOW TO CHECK FOR A VALID INPUT VOLTAGE
120 REM
130 IF V1 < 0 OR V1 >10.24 GOTO 1000
140 REM
150 REM NOW TO CALCULATE THE DIGITAL WORD FOR THE DAC
160 REM
170 X= V1/.040
180 REM
190 REM NOW TO ROUND OFF THE NUMBER
```

***Figure 9.14**—This program will calculate and output the digital value necessary for the DAC to produce a given voltage.*

```
200 REM
210 X=INT(X+.5)
220 REM
230 REM NOW TO OUTPUT THE WORD TO THE DAC
240 REM 38912,X
250 POKE 38912,X
260 REM
270 REM THE DAC VOLTAGE IS NOW SET. LOOP BACK TO START
280 REM
290 GOTO 60
800 REM
810 REM THIS SECTION PRINTS AN ERROR STATEMENT FOR A
820 REM VOLTAGE THAT WAS NOT BETWEEN 0.00 AND +10.24
830 REM
1000 PRINT
1010 PRINT"THE VOLTAGE "V1;" WAS NOT BETWEEN 0.00 AND 10.24 V"
1020 PRINT
1030 GOTO 80
```

*Figure 9.14 (continued).*

## 9.6 INCREASING THE OUTPUT DRIVE CAPABILITY OF THE DAC

If we want the DAC output voltage to drive a heavy current load, the current required may be greater than the rated output of the device. (See Figure 9.15.) The AD558 can output 5 milliamperes to a load with no electrical problems. In this section we will show how to increase the output current drive capability of the DAC from 5 milliamperes to several hundred milliamperes. In music applications the current required by the DAC is usually less than 5 milliamperes. For motor control applications, the current will be in the range from 100 to several hundred milliamperes.

One way to boost the output current is by using the circuit shown in Figure 9.16. In this circuit we take advantage of the current gain of

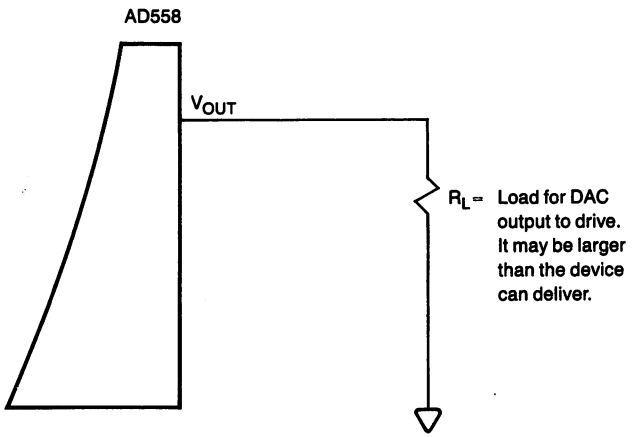



***Figure 9.15**–In some applications, the DAC output voltage will be required to drive a load that needs more output current than the DAC output pin can deliver. Usually, DACs will output only a few milliamperes of current to a load.*

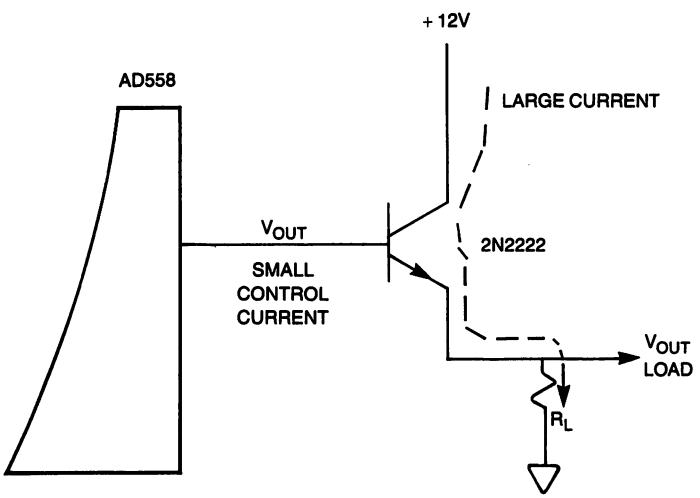



***Figure 9.16**–One way to increase the output drive current of the DAC is by using the circuit shown above. The heavy current is controlled by the smaller current in the base of the transistor. This base current is output by the DAC.*

a transistor, the 2N2222, manufactured by many companies. A disadvantage of this technique is that we do not know exactly what the output voltage at the load will be, because some voltage will drop across the base-emitter junction of the transistor (see Figure 9.17).

Another technique that can be used to increase the current drive capability of the AD558 is shown in Figure 9.18. This circuit makes use of an operational amplifier (or op amp) in addition to the transistor, to compensate for any voltage drop across the base-emitter junction. At this point we should briefly describe the function of an operational amplifer. The output of the op amp will automatically adjust itself so the two inputs are at the same voltage. This will allow the emitter of the transistor to be at the desired voltage potential, and let the transistor handle the heavy current. This circuit is a current amplifier with a voltage gain of 1. The voltage at the load in Figure 9.17 will be equal to the output voltage of the DAC.

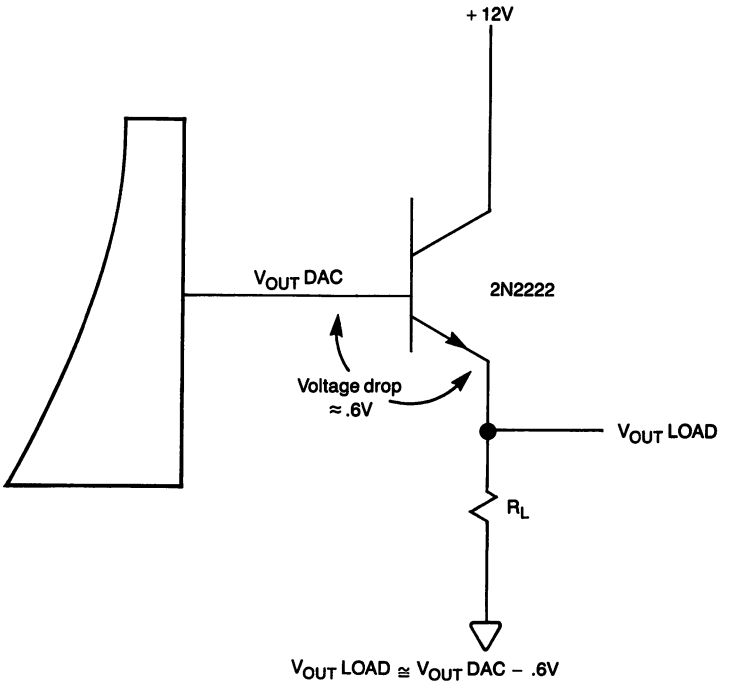



***Figure 9.17**–In the circuit of Figure 9.15, we do not know exactly what voltage will be delivered to the load. This is due to the voltage drop across the base-emitter junction of the transistor. The voltage drop is approximately .6 volts.*

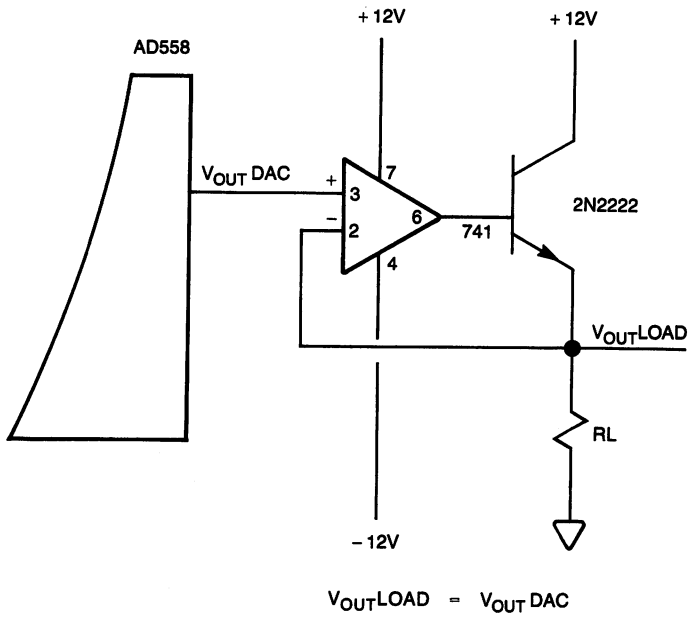



***Figure 9.18**–Schematic diagram of a circuit that will increase the output current drive of the DAC and compensate for the base-emitter voltage drop.*

## 9.7 SUMMARY

In this chapter we began by explaining the nature of digital-to-analog conversion (DAC). We showed that the number of unique output voltages of a converter depends on the number of digital input lines and the converter's range of output voltage levels. Also shown in this chapter was a general block diagram of a DAC. When you encounter a DAC for the first time, it will be helpful to keep in mind this general block diagram.

From this general discussion of the DAC, a formula was derived for calculating the minimum voltage change for a DAC with any number of digital inputs and any maximum output voltage swing. We then discussed how to determine what digital word to POKE to the DAC in order to set a desired output voltage.

The chapter finished by showing a sample program for controlling a DAC with the VIC-20 and giving general schematics for increasing the output drive capability of the device. Digital-to-analog converters are becoming quite popular in many home computer peripheral devices. If you understand the information given in this chapter, then using and controlling the peripheral devices that use DACs will be a much easier task.

Now that you have finished this text, the prospect of controlling external hardware with the VIC-20 computer should seem a much simpler problem than it did at first. We have presented and discussed many examples of interfacing the computer to peripheral equipment. Important timing and control signals used by the VIC-20 were clearly outlined, and you learned exactly how to use these signals. Throughout this text, the two prevailing concepts were those stated in Chapter 1. That is, computer control is made up of hardware and software that will:

1. Send electrical information to an external device, and
2. Receive electrical information from an external device.

At this point you know enough about controlling peripheral equipment with the VIC-20 to make your own VIC-20 connection. This will open the door to the applications described in this text as well as many others you may be able to imagine. Good luck and have fun making *the VIC-20 connection*.

## 9.8 FURTHER STUDY

In this text we have covered the basic topics involved in computer control with the VIC-20 computer. If you are interested in going deeper into the subject, there are other important topics to explore. (A good source to read for some of these topics is *Microprocessor Interfacing Techniques,* by R. Zaks and A. Lesea, Sybex, 3rd ed. 1979.)

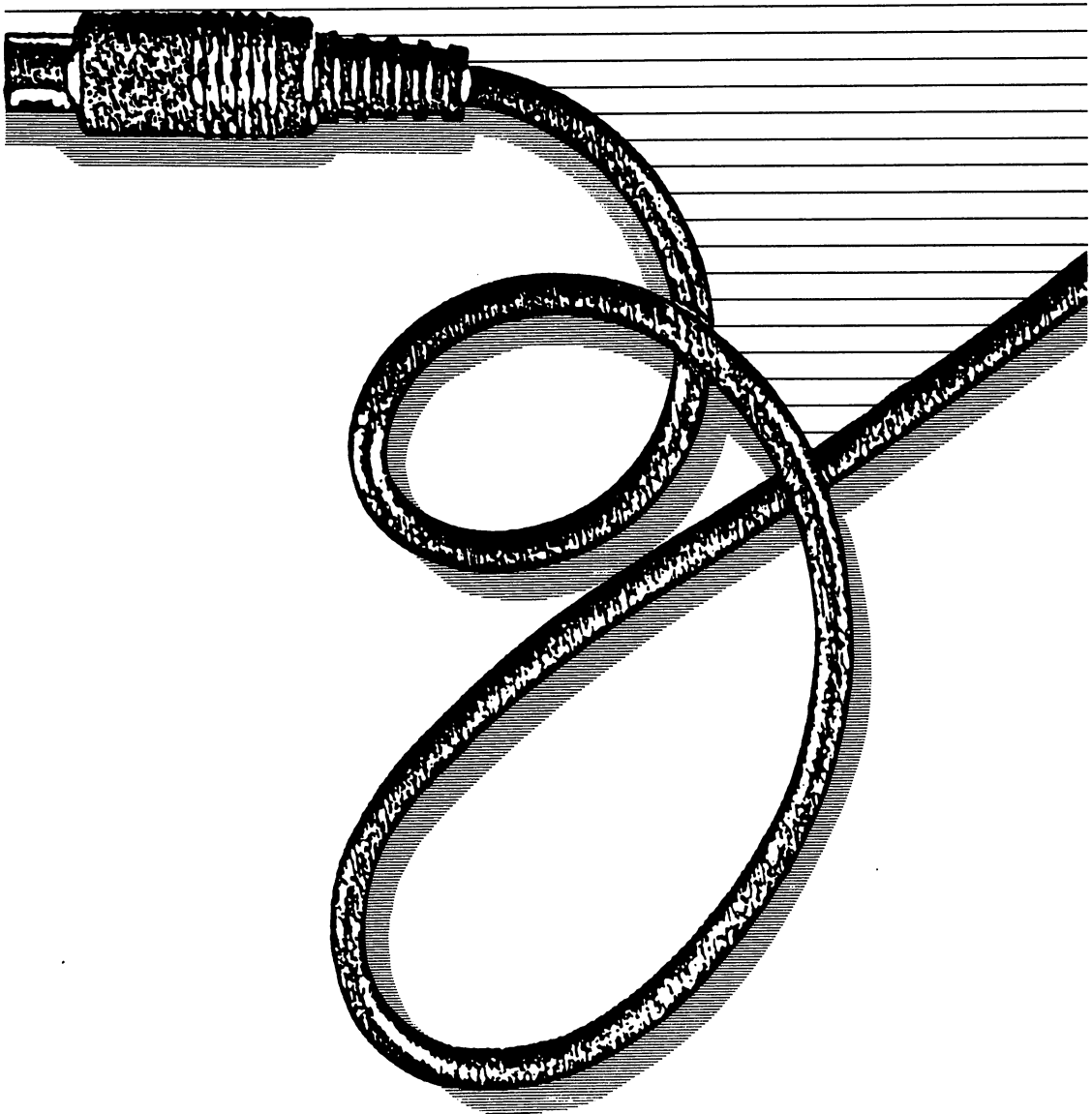
One area we have not discussed which is used often in computer interfacing is that of interrupts. Using interrupts, the peripheral device can request service from the computer only when necessary. At all other times, the computer will perform other tasks and will not service the external device at all.

Direct memory access (DMA) is another area that may be useful for further study. DMA is an electrical mode in which the computer's internal microprocessor is electrically removed from the system, allowing the peripheral device to control the system. In this way, the peripheral device can directly access the system's memory circuits without first going through the computer's central processing unit.

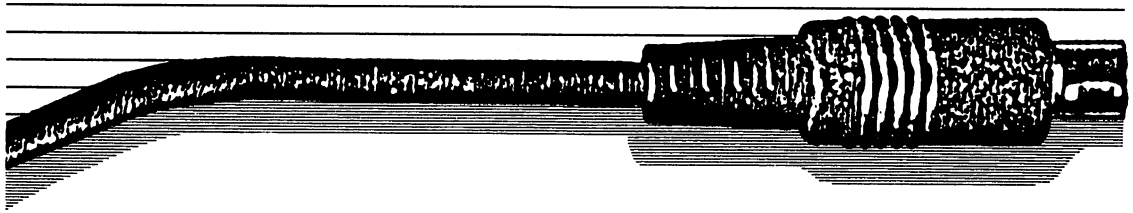
The last topic of study we should mention in the context of computer control is the use of different standard buses with external devices. Examples of these standard buses are IEEE-488 and RS-232. These buses will allow you to directly connect your computer to a peripheral device with no hardware modifications or special designs, which enables you to concentrate on software development.

These are some of the main topics you may want to study now that you are familiar with the essential elements of interfacing and computer control.

# MANUFACTURER'S DATA SHEETS



A



# 74LS00 SERIES INTEGRATED CIRCUITS
# LM135/LM235 TEMPERATURE SENSOR
# AD570 ANALOG-TO-DIGITAL CONVERTER
# AD558 DIGITAL-TO-ANALOG CONVERTER

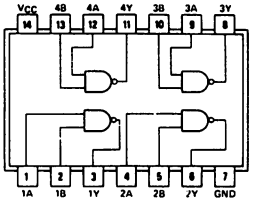
# 54/74 FAMILIES OF COMPATIBLE TTL CIRCUITS

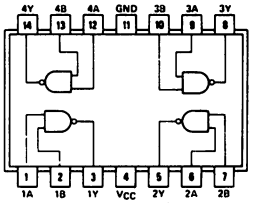
## PIN ASSIGNMENTS (TOP VIEWS)

**QUADRUPLE 2-INPUT POSITIVE-NAND GATES**

**00**

positive logic:
$Y = \overline{AB}$




SN5400 (J)
SN54H00 (J)
SN54L00 (J)
SN54LS00 (J, W)
SN54S00 (J, W)

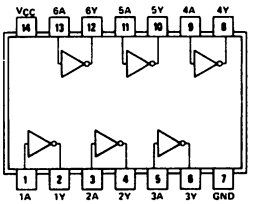
SN7400 (J, N)
SN74H00 (J, N)
SN74L00 (J, N)
SN74LS00 (J, N)
SN74S00 (J, N)

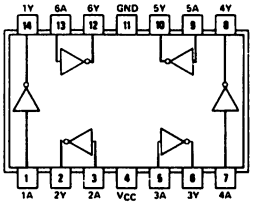
SN5400 (W)
SN54H00 (W)
SN54L00 (T)

**HEX INVERTERS**

**04**

positive logic:
$Y = \overline{A}$




SN5404 (J)
SN54H04 (J)
SN54L04 (J)
SN54LS04 (J, W)
SN54S04 (J, W)

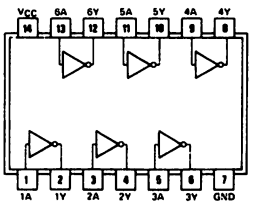
SN7404 (J, N)
SN74H04 (J, N)
SN74L04 (J, N)
SN74LS04 (J, N)
SN74S04 (J, N)

SN5404 (W)
SN54H04 (W)
SN54L04 (T)

**HEX INVERTER BUFFERS/DRIVERS WITH OPEN-COLLECTOR HIGH-VOLTAGE OUTPUTS**

**06**

positive logic:
$Y = \overline{A}$




SN5406 (J, W)  SN7406 (J, N)

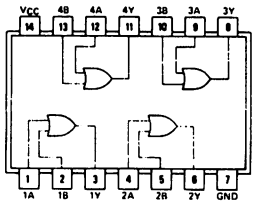
# 54/74 FAMILIES OF COMPATIBLE TTL CIRCUITS

## PIN ASSIGNMENTS (TOP VIEWS)

**QUADRUPLE 2-INPUT POSITIVE-OR GATES**

**32**

**positive logic:**
Y = A+B



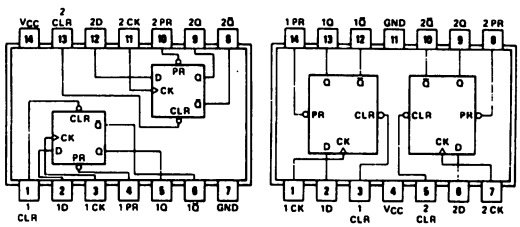

SN5432 (J, W) SN7432 (J, N)
SN54LS32 (J, W) SN74LS32 (J, N)
SN54S32 (J, W) SN74S32 (J, N)

**DUAL D-TYPE POSITIVE-EDGE-TRIGGERED FLIP-FLOPS WITH PRESET AND CLEAR**

**74**

**FUNCTION TABLE**

| INPUTS | | | | OUTPUTS | |
|---|---|---|---|---|---|
| PRESET | CLEAR | CLOCK | D | Q | $\bar{Q}$ |
| L | H | X | X | H | L |
| H | L | X | X | L | H |
| L | L | X | X | H* | H* |
| H | H | ↑ | H | H | L |
| H | H | ↑ | L | L | H |
| H | H | L | X | $Q_0$ | $\bar{Q}_0$ |

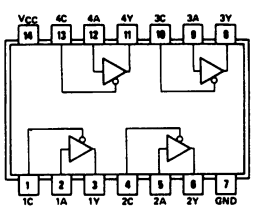



SN5474 (J) SN7474 (J, N) SN5474 (W)
SN54H74 (J) SN74H74 (J, N) SN54H74 (W)
SN54L74 (J) SN74L74 (J, N) SN54L74 (T)
SN54LS74A (J, W) SN74LS74A (J, N)
SN54S74 (J, W) SN74S74 (J, N)

**QUADRUPLE BUS BUFFER GATES WITH THREE-STATE OUTPUTS**

**125**

**positive logic:**
Y = A
Output is off (disabled) when C is high.



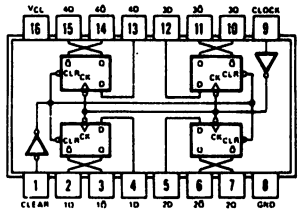

SN54125 (J, W) SN74125 (J, N)
SN54LS125A (J, W) SN74LS125A (J, N)

# 54/74 FAMILIES OF COMPATIBLE TTL CIRCUITS

## PIN ASSIGNMENTS (TOP VIEWS)

QUAD D-TYPE FLIP-FLOPS

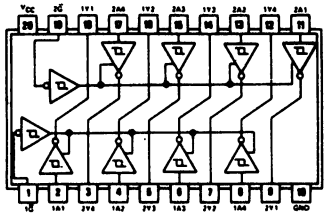
**175** COMPLEMENTARY OUTPUTS
COMMON DIRECT CLEAR



SN54175 (J, W) SN74175 (J, N)
SN54LS175 (J, W) SN74LS175 (J, N)
SN54S175 (J, W) SN74S175 (J, N)

OCTAL BUFFERS/LINE DRIVERS/LINE RECEIVERS

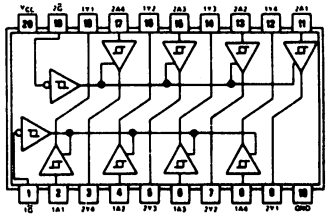
**240** INVERTED 3-STATE OUTPUTS



SN54LS240 (J) SN74LS240 (J, N)
SN54S240 (J) SN74S240 (J, N)

OCTAL BUFFERS/LINE DRIVERS/LINE RECEIVERS

**244** NONINVERTED 3-STATE OUTPUTS



SN54LS244 (J) SN74LS244 (J, N)

National Semiconductor

November 1980

# LM135/LM235/LM335, LM135A/LM235A/LM335A Precision Temperature Sensors

## General Description

The LM135 series are precision, easily-calibrated, integrated circuit temperature sensors. Operating as a 2-terminal zener, the LM135 has a breakdown voltage directly proportional to absolute temperature at +10 mV/°K. With less than 1Ω dynamic impedance the device operates over a current range of 400 μA to 5 mA with virtually no change in performance. When calibrated at 25°C the LM135 has typically less than 1°C error over a 100°C temperature range. Unlike other sensors the LM135 has a linear output.

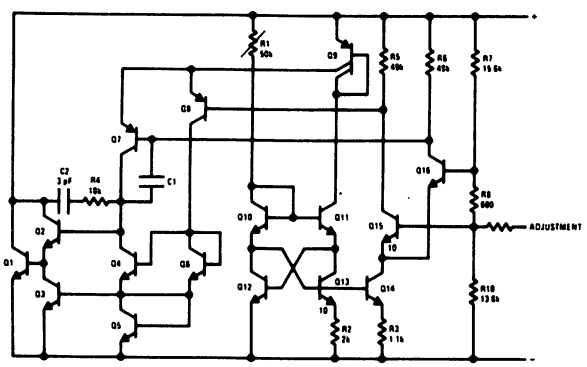
Applications for the LM135 include almost any type of temperature sensing over a −55°C to +150°C temperature range. The low impedance and linear output make interfacing to readout or control circuitry especially easy.

The LM135 operates over a −55°C to +150°C temperature range while the LM235 operates over a −40°C to +125°C temperature range. The LM335 operates from −40°C to +100°C. The LM135/LM235/LM335 are available packaged in hermetic TO-46 transistor packages while the LM335 is also available in plastic TO-92 packages.

## Features

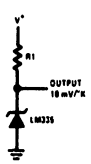
- Directly calibrated in °Kelvin
- 1°C initial accuracy available
- Operates from 400 μA to 5 mA
- Less than 1Ω dynamic impedance
- Easily calibrated
- Wide operating temperature range
- 200°C overrange
- Low cost

## Schematic Diagram

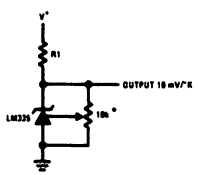



## Typical Applications

**Basic Temperature Sensor**

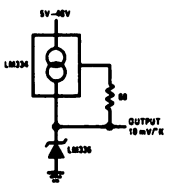



**Calibrated Sensor**




*Calibrate for 2.982V at 25°C

**Wide Operating Supply**



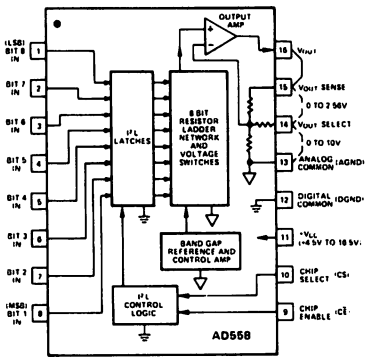
ANALOG DEVICES

# DACPORT™ Low Cost Complete μP-Compatible 8-Bit DAC

# AD558*

**FEATURES**
**Complete 8-Bit DAC**
**Voltage Output – 2 Calibrated Ranges**
**Internal Precision Band-Gap Reference**
**Single-Supply Operation: +5V to +15V**
**Full Microprocessor Interface**
**Fast: 1μs Voltage Settling to ±1/2LSB**
**Low Power: 75mW**
**No User Trims**
**Guaranteed Monotonic Over Temperature**
**All Errors Specified $T_{min}$ to $T_{max}$**
**Small 16-Pin DIP Package**
**Single Laser-Wafer-Trimmed Chip for Hybrids**
**Low Cost**

**AD558 FUNCTIONAL BLOCK DIAGRAM**




**TO-116**

**PRODUCT DESCRIPTION**

The AD558 DACPORT is a complete voltage-output 8-bit digital-to-analog converter, including output amplifier, full microprocessor interface and precision voltage reference on a single monolithic chip. No external components or trims are required to interface, with full accuracy, an 8-bit data bus to an analog system.

The performance and versatility of the DACPORT is a result of several recently-developed monolithic bipolar technologies. The complete microprocessor interface and control logic is implemented with integrated injection logic ($I^2L$), an extremely dense and low-power logic structure that is process-compatible with linear bipolar fabrication. The internal precision voltage reference is the patented low-voltage band-gap circuit which permits full-accuracy performance on a single +5V to +15V power supply. Thin-film silicon-chromium resistors provide the stability required for guaranteed monotonic operation over the entire operating temperature range (all grades), while recent advances in laser-wafer-trimming of these thin-film resistors permit absolute calibration at the factory to within ±1LSB; thus no user-trims for gain or offset are required. A new circuit design provides voltage settling to ±1/2LSB for a full-scale step in 800ns.

The AD558 is available in four performance grades. The AD558J and K are specified for use over the 0 to +70°C temperature range, while the AD558S and T grades are specified for -55°C to +125°C operation. The hermetically-sealed ceramic package is standard. Processing to MIL-STD-883, Class B is optional on S and T grades.

**PRODUCT HIGHLIGHTS**

1. The 8-bit $I^2L$ input register and fully microprocessor-compatible control logic allow the AD558 to be directly connected to 8- or 16-bit data buses and operated with standard control signals. The latch may be disabled for direct DAC interfacing.
2. The laser-trimmed on-chip SiCr thin-film resistors are calibrated for absolute accuracy and linearity at the factory. Therefore, no user trims are necessary for full rated accuracy over the operating temperature range.
3. The inclusion of a precision low-voltage band-gap reference eliminates the need to specify and apply a separate reference source.
4. The voltage-switching structure of the AD558 DAC section along with a high-speed output amplifier and laser-trimmed resistors give the user a choice of 0V to +2.56V or 0V to +10V output ranges, selectable by pin-strapping. Circuitry is internally compensated for minimum settling time on both ranges; typically settling to ±1/2LSB for a full-scale 2.55 volt step in 800ns.
5. The AD558 is designed and specified to operate from a single +4.5V to +16.5V power supply.
6. Low digital input currents, 100μA max, minimize bus loading. Input thresholds are TTL/low voltage CMOS compatible over the entire operating $V_{CC}$ range.
7. The single-chip, low power $I^2L$ design of the AD558 is inherently more reliable than hybrid multi-chip or conventional single-chip bipolar designs. The AD558S and T grades, which are specified over the -55°C to +125°C temperature range, are available processed to MIL-STD-883, Class B.
8. All AD558 grades are available in chip form with guaranteed specifications from +25°C to $T_{max}$. MIL-STD-883, Class B visual inspection is standard on Analog Devices bipolar chips. Contact the factory for additional chip information.

# SPECIFICATIONS (typical @ $T_A$ = +25°C, $V_{CC}$ = +5V to +15V unless otherwise specified)

| MODEL | AD558J | AD558K | AD558S[1] | AD558T[1] |
|---|---|---|---|---|
| RESOLUTION | 8 Bits | • | • | • |
| RELATIVE ACCURACY[2] | | | | |
| 0 to +70°C | ±1/2LSB max | ±1/4LSB max | • | •• |
| -55°C to +125°C | – | – | ±3/4LSB max | ±3/8LSB max |
| OUTPUT | | | | |
| Ranges | 0V to +2.56V | • | • | • |
| | 0V to +10V [3] | • | • | • |
| Current, Source | +5mA | • | +5mA min | ••• |
| Sink | Internal Passive Pull-Down to Ground[4] | • | • | • |
| OUTPUT SETTLING TIME[5] | | | | |
| 0 to 2.56 volt range | 0.8μs (1.5μs max) | • | • | • |
| 0 to 10 volt range[3] | 2.0μs (3.0μs max) | • | • | • |
| FULL SCALE ACCURACY | | | | |
| @ 25°C | ±1.5LSB (±0.6%) max | ±0.5LSB (±0.2%) max | • | •• |
| $T_{min}$ to $T_{max}$ | ±2.5LSB (±1.0%) max | ±1LSB (±0.4%) max | • | •• |
| ZERO ERROR | | | | |
| @ 25°C | ±1LSB max | ±1/2LSB max | • | •• |
| $T_{min}$ to $T_{max}$ | ±2LSB max | ±1LSB max | • | •• |
| MONOTONICITY[6] | | | | |
| $T_{min}$ to $T_{max}$ | Guaranteed | • | • | • |
| DIGITAL INPUTS | | | | |
| $T_{min}$ to $T_{max}$ | | | | |
| Input Current | ±100μA max | • | • | • |
| Data Inputs, Voltage | | | | |
| Bit On – Logic "1" | 2.0V min | • | • | • |
| Bit Off – Logic "0" | 0.8V max | • | • | • |
| Control Inputs, Voltage | | | | |
| On – Logic "1" | 2.0V min | • | • | • |
| Off – Logic "0" | 0.8V max | • | • | • |
| Input Capacitance | 4pF | • | • | • |
| TIMING[7] | | | | |
| $T_{min}$ to $T_{max}$ | | | | |
| $t_W$ (Strobe Pulse Width) | 100ns min | • | • | • |
| $t_{DH}$ (Data Hold Time) | 10ns min | • | • | • |
| $t_{DS}$ (Data Set-Up Time) | 100ns min | • | • | • |
| POWER SUPPLY | | | | |
| Operating Voltage Range ($V_{CC}$) | | | | |
| 2.56 Volt Range | +4.5V to +16.5V | • | • | • |
| 10 Volt Range | +11.4V to +16.5V | • | • | • |
| Current ($I_{CC}$) | 15mA typ, 25mA max | • | • | • |
| Rejection Ratio | 0.03%/% max | • | • | • |
| POWER DISSIPATION, $V_{CC}$ = 5V | 75mW (125mW max) | • | • | • |
| $V_{CC}$ = 15V | 225mW (375mW max) | • | • | • |
| OPERATING TEMPERATURE RANGE | | | | |
| $T_{min}$ | 0°C | • | -55°C | ••• |
| $T_{max}$ | +70°C | • | +125°C | ••• |

**ABSOLUTE MAXIMUM RATINGS**

$V_{CC}$ to Ground . . . . . .0V to +18V
Digital Inputs (Pins 1-10) . . . . . 0 to +7.0V
$V_{OUT}$ . . . . . Indefinite Short to Ground
Momentary Short to $V_{CC}$
Power Dissipation . . . . . 450mW
Storage Temperature Range
D (ceramic) Package. . . . . -55°C to +150°C
Lead Temperature (soldering, 10 second). . . . . 300°C
Thermal Resistance
Junction to Ambient/Junction to Case
D (ceramic) Package. . . . . .100/30°C/W

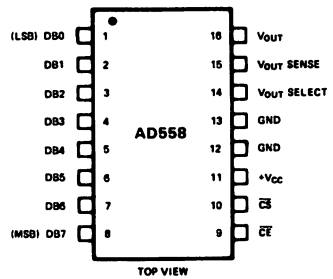



*Figure 1. AD558 Pin Configuration*

**AD558 ORDERING GUIDE**

| Model | Package | Temperature | Relative Accuracy Error Max $T_{min}$ to $T_{max}$ | Full-Scale Error, Max $T_{min}$ to $T_{max}$ | Package Style[1] |
|---|---|---|---|---|---|
| AD558JN | Plastic | 0 to +70°C | ±1/2LSB | ±2.5LSB | N16A[2] |
| AD558KN | Plastic | 0 to +70°C | ±1/4LSB | ±1LSB | N16A[2] |
| AD558JD | Ceramic | 0 to +70°C | ±1/2LSB | ±2.5LSB | D16A |
| AD558KD | Ceramic | 0 to +70°C | ±1/4LSB | ±1LSB | D16A |
| AD558SD | Ceramic | -55°C to +125°C | ±3/4LSB | ±2.5LSB | D16A |
| AD558SD/883B | Ceramic | -55°C to +125°C | ±3/4LSB | ±2.5LSB | D16A |
| AD558TD | Ceramic | -55°C to +125°C | ±3/8LSB | ±1LSB | D16A |
| AD558TD/883B | Ceramic | -55°C to +125°C | ±3/8LSB | ±1LSB | D16A |

[1] See Section 20 for package outline information.
[2] To be available June, 1982.

**CIRCUIT DESCRIPTION**

The AD558 consists of four major functional blocks, fabricated on a single monolithic chip (see Figure 2). The main D to A converter section uses eight equally-weighted laser-trimmed current sources switched into a silicon-chromium thin-film R/2R resistor ladder network to give a direct but unbuffered 0mV to 400mV output range. The transistors that form the DAC switches are PNPs; this allows direct positive-voltage logic interface and a zero-based output range.

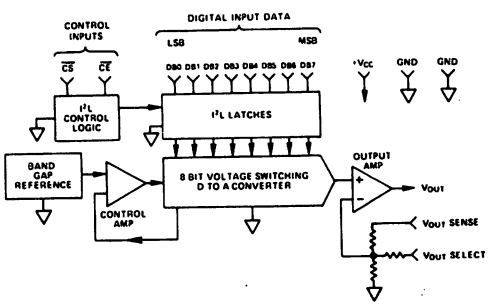



*Figure 2. AD558 Functional Block Diagram*

The high-speed output buffer amplifier is operated in the non-inverting mode with gain determined by the user-connections at the output range select pin. The gain-setting application resistors are thin-film laser-trimmed to match and track the DAC resistors and to assure precise initial calibration of the two output ranges, 0V to 2.56V and 0V to 10V. The amplifier output stage is an NPN transistor with passive pull-down for zero-based output capability with a single power supply.

The internal precision voltage reference is of the patented band-gap type. This design produces a reference voltage of 1.2 volts and thus, unlike 6.3 volt temperature-compensated zeners, may be operated from a single, low-voltage logic power supply. The microprocessor interface logic consists of an 8-bit data latch and control circuitry. Low-power, small geometry and high-speed are advantages of the $I^2L$ design as applied to this section. $I^2L$ is bipolar process compatible so that the performance of the analog sections need not be compromised to provide on-chip logic capabilities. The control logic allows the latches to be operated from a decoded microprocessor address and write signal. If the application does not involve a $\mu$P or data bus, wiring $\overline{CS}$ and $\overline{CE}$ to ground renders the latches "transparent" for direct DAC access.

# AD558 Applications

## CONNECTING THE AD558

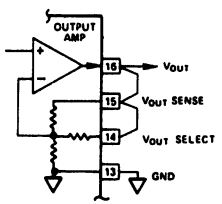
The AD558 has been configured for ease of application. All reference, output amplifier and logic connections are made internally. In addition, all calibration trims are performed at the factory assuring specified accuracy without user trims. The only connection decision that must be made by the user is a single jumper to select output voltage range. Clean circuit-board layout is facilitated by isolating all digital bit inputs on one side of the package; analog outputs are on the opposite side.

Figure 3 shows the two alternative output range connections. The 0V to 2.56V range may be selected for use with any power supply between +4.5V and +16.5V. The 0V to 10V range requires a power supply of +11.4V to +16.5V.

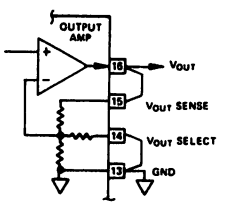
Because of its precise factory calibration, the AD558 is intended to be operated without user trims for gain and offset; therefore no provisions have been made for such user-trims. If a small increase in scale is required, however, it may be accomplished by slightly altering the effective gain of the output buffer. A resistor in series with $V_{OUT}$ SENSE will increase the output range.

For example if a 0V to 10.24V output range is desired (40mV = 1LSB), a nominal resistance of 850Ω is required. It must be remembered that, although the internal resistors all ratio-match and track, the *absolute* tolerance of these resistors is typically ±20% and the *absolute* TC is typically -50ppm/°C (0 to -100ppm/°C). That must be considered when re-scaling is performed. Figure 4 shows the recommended circuitry for a full-scale output range of 10.24 volts. Internal resistance values shown are nominal.

*NOTE: Decreasing the scale by putting a resistor in series with GND will not work properly due to the code-dependent currents in GND. Adjusting offset by injecting dc at GND is not recommended for the same reason.*




*a. 0V to 2.56V Output Range*




*b. 0V to 10V Output Range*

***Figure 3. Connection Diagrams***

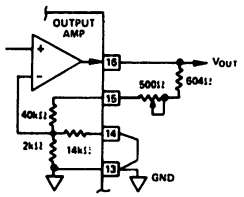



***Figure 4. 10.24V Full-Scale Connection***

## GROUNDING AND BYPASSING*

All precision converter products require careful application of good grounding practices to maintain full rated performance. Because the AD558 is intended for application in microcomputer systems where digital noise is prevalent, special care must be taken to assure that its inherent precision is realized.

The AD558 has two ground (common) pins; this minimizes ground drops and noise in the analog signal path. Figure 5 shows how the ground connections should be made.

It is often advisable to maintain separate analog and digital grounds throughout a complete system, tying them common in one place only. If the common tie-point is remote and accidental disconnection of that one common tie-point occurs due to card removal with power on, a large differential voltage between the two commons could develop. To protect devices that interface to both digital and analog parts of the system, such as the AD558, it is recommended that common ground tie-points should be provided at *each* such device. If only one system ground can be connected directly to the AD558, it is recommended that analog common be selected.

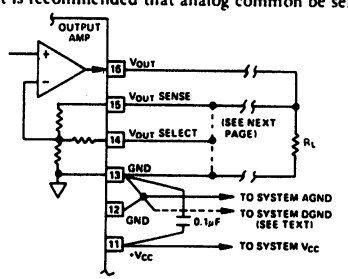



***Figure 5. Recommended Grounding and Bypassing***

## POWER SUPPLY CONSIDERATIONS

The AD558 is designed to operate from a single positive power supply voltage. Specified performance is achieved for any supply voltage between +4.5V and +16.5V. This makes the AD558 ideal for battery-operated, portable, automotive or digital main-frame applications.

The only consideration in selecting a supply voltage is that, in order to be able to use the 0V to 10V output range, the power supply voltage must be between +11.4V and +16.5V. If, however, the 0V to 2.56V range is to be used, power consumption will be minimized by utilizing the lowest available supply voltage (above +4.5V).

**TIMING AND CONTROL**

The AD558 has data input latches that simplify interface to 8- and 16-bit data buses. These latches are controlled by Chip Enable ($\overline{CE}$) and Chip Select ($\overline{CS}$) inputs, pins 9 and 10 respectively. $\overline{CE}$ and $\overline{CS}$ are internally "NORed" so that the latches transmit input data to the DAC section when both $\overline{CE}$ and $\overline{CS}$ are at Logic "0". If the application does not involve a data bus, a "00" condition allows for direct operation of the DAC. When either $\overline{CE}$ or $\overline{CS}$ go to Logic "1", the input data is latched into the registers and held until both $\overline{CE}$ and $\overline{CS}$ return to "0". (Unused $\overline{CE}$ or $\overline{CS}$ inputs should be tied to ground.) The truth table is given in Table I. The logic function is also shown in Figure 6.

| Input Data | $\overline{CE}$ | $\overline{CS}$ | DAC Data | Latch Condition |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | "transparent" |
| 1 | 0 | 0 | 1 | "transparent" |
| 0 | ∫ | 0 | 0 | latching |
| 1 | ∫ | 0 | 1 | latching |
| 0 | 0 | ∫ | 0 | latching |
| 1 | 0 | ∫ | 1 | latching |
| X | 1 | X | previous data | latched |
| X | X | 1 | previous data | latched |

Notes: X = Does not matter
∫ = Logic Threshold at Positive-Going Transition

*Table I. AD558 Control Logic Truth Table*

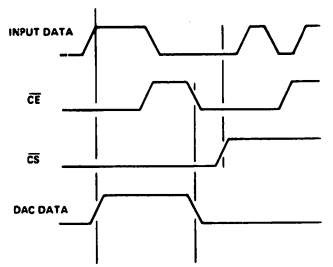



*Figure 6. AD558 Control Logic Function*

Figure 7 shows the timing for the data and control signals; $\overline{CE}$ and $\overline{CS}$ are identical in timing as well as in function.

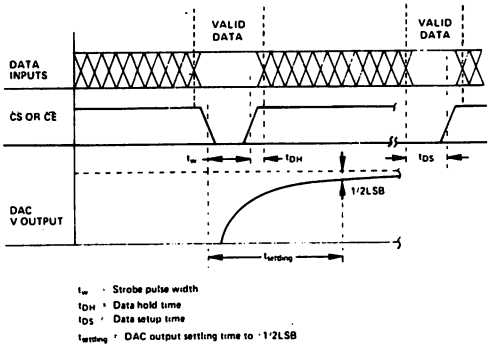
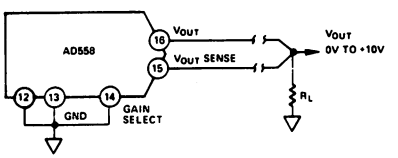


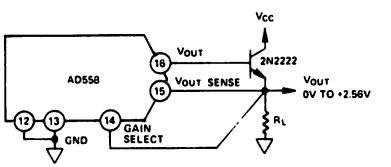

*Figure 7. AD558 Timing*

**USE OF $V_{OUT}$ SENSE**

Separate access to the feedback resistor of the output amplifier allows additional application versatility. Figure 8a shows how I X R drops in long lines to remote loads may be cancelled by putting the drops "inside the loop". Figure 8b shows how the separate sense may be used to provide a higher output current by feeding back around a simple current booster.




*a. Compensation for I x R Drops in Output Lines*




*b. Output Current Booster*

*Figure 8. Use of $V_{OUT}$ Sense*

# Applying the AD558

### OPTIMIZING SETTLING TIME

In order to provide single-supply operation and zero-based output voltage ranges, the AD558 output stage has a passive "pull-down" to ground. As a result, settling time for negative-going output steps may be longer than for positive-going output steps. The relative difference depends on load resistance and capacitance. If a negative power supply is available, the negative-going settling time may be improved by adding a pull-down resistor from the output to the negative supply as shown in Figure 9. The value of the resistor should be such that, at zero voltage out, current through that resistor is 0.5mA max.

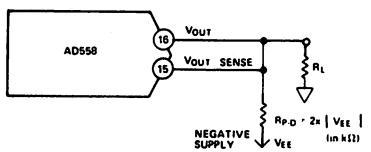



*Figure 9. Improved Settling Time*

### BIPOLAR OUTPUT RANGES

The AD558 was designed for operation from a single power supply and is thus capable of providing only unipolar (0V to +2.56 and 0V to 10V) output ranges. If a negative supply is available, bipolar output ranges may be achieved by suitable output offsetting and scaling. Figure 10 shows how a ±1.28 volt output range may be achieved when a -5 volt power supply is available. The offset is provided by the AD589 precision 1.2 volt reference which will operate from a +5 volt supply. The AD544 output amplifier can provide the necessary ±1.28 volt output swing from ±5 volt supplies. Coding is complementary offset binary.

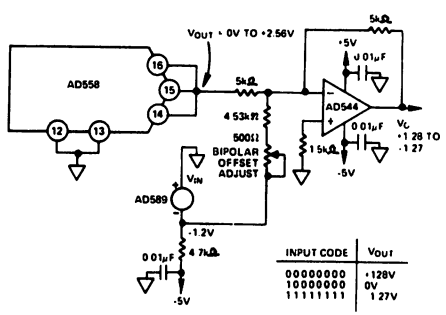
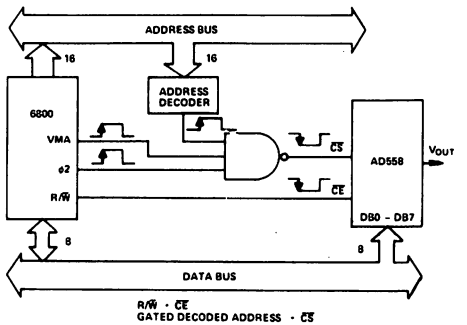


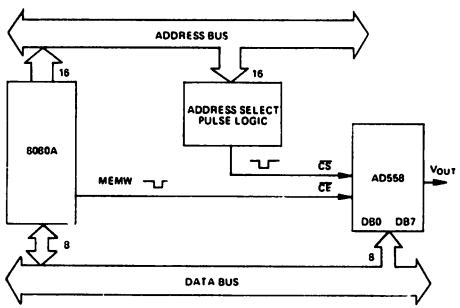

*Figure 10. Bipolar Operation of AD558 from ±5V Supplies*

### INTERFACING THE AD558 TO MICROPROCESSOR DATA BUSES*

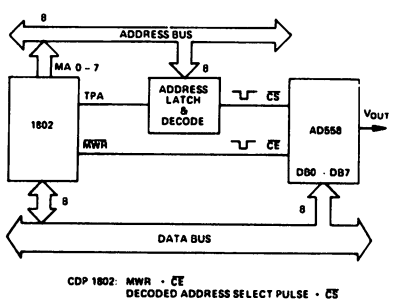
The AD558 is configured to act like a "write only" location in memory that may be made to coincide with a read only memory location or with a RAM location. The latter case allows data previously written into the DAC to be read back later via the RAM. Address decoding is partially complete for either ROM or RAM. Figure 11 shows interfaces for three popular microprocessor systems.




*a. 6800/AD558 Interface*




*b. 8080A/AD558 Interface*




*c. 1802/AD558 Interface*

*Figure 11. Interfacing the AD558 to Microprocessors*

*The microprocessor-interface capabilities of the AD558 are extensive. A comprehensive application note, "Interfacing the AD558 DACPORT™ to Microprocessors" is available from any Analog Devices Sales Office upon request, free of charge.

## AD558 Performance (typical @ +25°C, $V_{CC}$ = +5V to +15V unless otherwise noted)

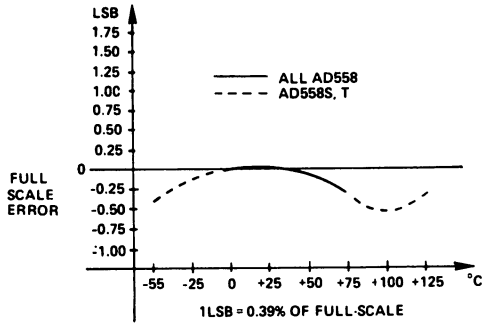



*Figure 12. Full Scale Accuracy vs. Temperature Performance of AD558*

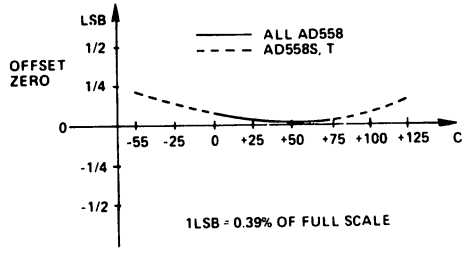



*Figure 13. Zero Drift vs. Temperature Performance of AD558*

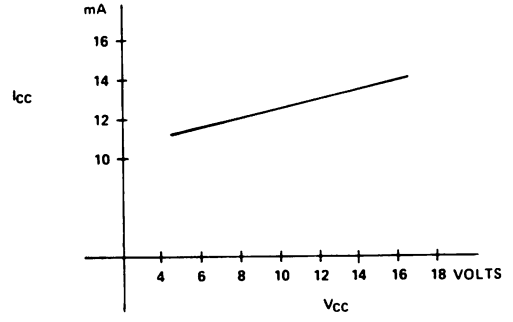



*Figure 14. Quiescent Current vs. Power Supply Voltage for AD558*

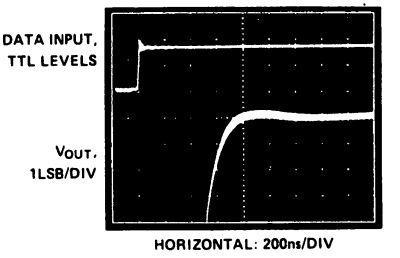



*Figure 15. AD558 Settling Characteristic Detai 0V to 2.56V Output Range Full-Scale Step*

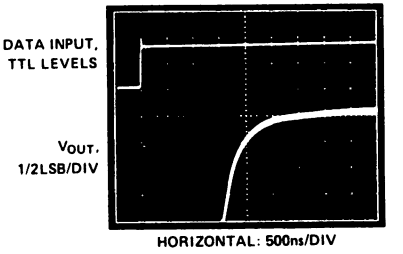



*Figure 16. AD558 Settling Chracteristic Detail 0V to 10V Output Range Full-Scale Step*

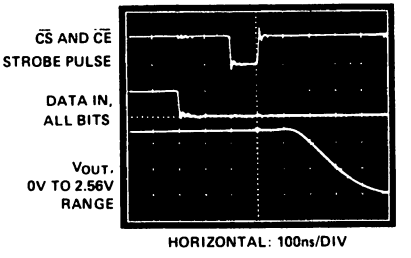



*Figure 17. AD558 Logic Timing*

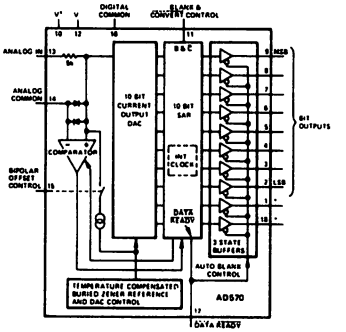
ANALOG DEVICES

# Low Cost, Complete IC 8-Bit A to D Converter

# AD570*

**FEATURES**
**Complete A/D Converter with Reference and Clock**
**Fast Successive Approximation Conversion – 25μs**
**No Missing Codes Over Temperature**
**0 to +70°C – AD570J**
**-55°C to +125°C – AD570S**
**Digital Multiplexing – 3 State Outputs**
**18-Pin DIP**
**Low Cost Monolithic Construction**

**AD570 FUNCTIONAL BLOCK DIAGRAM**




**18-PIN DUAL IN LINE PACKAGE**

**PRODUCT DESCRIPTION**

The AD570 is an 8-bit successive approximation A/D converter consisting of a DAC, voltage reference, clock, comparator, successive approximation register and output buffers – all fabricated on a single chip. No external components are required to perform a full accuracy 8-bit conversion in 25μs.

The AD570 incorporates the most advanced integrated circuit design and processing technology available today. $I^2L$ (integrated injection logic) processing in the fabrication of the SAR function along with laser trimming of the high stability SiCr thin film resistor ladder network at the wafer stage (LWT) and a temperature compensated, subsurface Zener reference insures full 8-bit accuracy at low cost.

Operating on supplies of +5V and -15V, the AD570 will accept analog inputs of 0 to +10V unipolar or ±5V bipolar, externally selectable. As the BLANK and $\overline{\text{CONVERT}}$ input is driven low, the three state outputs will be open and a conversion will commence. Upon completion of the conversion, the $\overline{\text{DATA READY}}$ line will go low and the data will appear at the output. Pulling the BLANK and $\overline{\text{CONVERT}}$ input high blanks the outputs and readies the device for the next conversion. The AD570 executes a true 8-bit conversion with no missing codes in approximately 25μs.

The AD570 is available in two versions; the AD570J is specified for the 0 to 70°C temperature range, the AD570S for -55°C to +125°C. Both guarantee full 8-bit accuracy and no missing codes over their respective temperature ranges. The AD570J is also offered in an 18-pin plastic DIP.

**PRODUCT HIGHLIGHTS**

1. The AD570 is a complete 8-bit A/D converter. No external components are required to perform a conversion. Full scale calibration accuracy of ±0.8% (2LSB of 8 bits) is achieved without external trims.
2. The AD570 is a single chip device employing the most advanced IC processing techniques. Thus, the user has at his disposal a truly precision component with the reliability and low cost inherent in monolithic construction.
3. The AD570 accepts either unipolar (0 to +10V) or bipolar (-5V to +5V) analog inputs by simply grounding or opening a single pin.
4. The device offers true 8-bit accuracy and exhibits no missing codes over its entire operating temperature range.
5. Operation is guaranteed with -15V and +5V supplies. The device will also operate with a -12V supply.
6. The AD570S is also available with processing to MIL-STD-883, Class B. The single chip construction and functional completeness make the AD570 especially attractive for high reliability applications.

*Protected by Patent Nos. 3940760, 4213806 and 4136349.

# SPECIFICATIONS

(typical @ +25°C with V+ = +5V, V- = -15V, all voltages measured with respect to digital common, unless otherwise indicated)

| MODEL | AD570J | AD570S[1] |
|---|---|---|
| RESOLUTION[2] | 8 Bits | • |
| RELATIVE ACCURACY @ 25°C[2,3,4] | ±1/2LSB max | • |
| $T_{min}$ to $T_{max}$ | ±1/2LSB max | • |
| FULL SCALE CALIBRATION[4,5] (With 15Ω Resistor In Series With Analog Input | ±2LSB (typ) | • |
| UNIPOLAR OFFSET (max)[4] | ±1/2LSB | • |
| BIPOLAR OFFSET (max)[4] | ±1/2LSB | • |
| DIFFERENTIAL NONLINEARITY (Resolution for Which no Missing Codes are Guaranteed) | | |
| +25°C | 8 Bits | • |
| $T_{min}$ to $T_{max}$ | 8 Bits | • |
| TEMPERATURE RANGE | 0 to +70°C | -55°C to +125°C |
| TEMPERATURE COEFFICIENTS[4] | | |
| Guaranteed max Change | | |
| $T_{min}$ to $T_{max}$ | | |
| Unipolar Offset | ±1LSB (88ppm/°C) | ±1LSB (40ppm/°C) |
| Bipolar Offset | ±1LSB (88ppm/°C) | ±1LSB (40ppm/°C) |
| Full Scale Calibration[6] (With 15Ω Fixed Resistor or 200Ω Trimmer) | ±2LSB (176ppm/°C) | ±2LSB (80ppm/°C) |
| POWER SUPPLY REJECTION[4] | | |
| Max Change In Full Scale Calibration | | |
| TTL Positive Supply | | |
| +4.5V≤V+≤+5.5V | ±2LSB max | • |
| Negative Supply | | |
| -16.0V≤V-≤-13.5V | ±2LSB max | • |
| ANALOG INPUT RESISTANCE | 3kΩ min | • |
| | 5kΩ typ | • |
| | 7kΩ max | • |
| ANALOG INPUT RANGES (Analog Input to Analog Common) | | |
| Unipolar | 0 to +10V | • |
| Bipolar | -5V to +5V | • |
| OUTPUT CODING | | |
| Unipolar | Positive True Binary | • |
| Bipolar | Positive True Offset Binary | • |
| LOGIC OUTPUT | | |
| Bit Outputs and $\overline{\text{Data Ready}}$ | | |
| Output Sink Current | 3.2mA min | • |
| ($V_{OUT}$ = 0.4V max, $T_{min}$ to $T_{max}$) | (2TTL Loads) | • |
| Output Source Current (Bit Outputs)[7] | | |
| ($V_{OUT}$ = 2.4V min, $T_{min}$ to $T_{max}$) | 0.5mA min | • |
| Output Leakage When Blanked | ±40μA max | • |
| LOGIC INPUT | | |
| Blank and Convert Input | | |
| $0 \leq V_{in} \leq V+$ | ±40μA max | • |
| Blank – Logic "1" | 2.0V min | • |
| Convert – Logic "0" | 0.8V max | • |
| CONVERSION TIME | 15μs min | • |
| | 25μs typ | • |
| | 40μs max | • |

| ALL MODELS | |
|---|---|
| **POWER SUPPLY** | |
| Absolute Maximum | |
| V+ | +7V |
| V- | -16.5V |
| Specified Operating – Rated Performance | |
| V+ | +5V |
| V- | -15V |
| Operating Range | |
| V+ | +4.5V to +5.5V |
| V- | -12.0V to -16.5V |
| Operating Current | |
| Blank Mode | |
| V+ = +5V | 2mA typ (10mA max) |
| V- = -15V | 9mA typ (15mA max) |
| Convert Mode | |
| V+ = +5V | 5mA |
| V- = -15V | 10mA |

*Specifications same as AD570J

Specifications subject to change without notice.

NOTES

[1] The AD570S is available processed and screened to the requirements of MIL-STD-883B, Class B. When ordering, specify the AD570SD/883B.

[2] The AD570 is a selected version of the AD571 10-bit A to D converter. As such, some devices may exhibit 9 or 10 bits of relative accuracy or resolution, but that is neither tested nor guaranteed. Only TTL logic inputs should be connected to pins 1 and 18 (or no connection made) or damage may result.

[3] Relative accuracy is defined as the deviation of the code transition points from the ideal transfer point on a straight line from the zero to the full scale of the device.

[4] Specifications given in LSB's refer to the weight of a least significant bit at the 8-bit level, which is 0.39% of full-scale.

[5] Full scale calibration is guaranteed trimmable to zero with an external 200Ω potentiometer in place of the 15Ω fixed resistor. Full scale is defined as 10 volts minus 1 LSB, or 9.961 volts.

[6] Full Scale Calibration Temperature Coefficient includes effects of unipolar offset drift as well as gain drift.

[7] The Data output lines have active pull-ups to source 0.5mA. The $\overline{\text{DATA READY}}$ line is open collector with a nominal 6kΩ internal pull-up resistor.

**ABSOLUTE MAXIMUM RATINGS**

V+ to Digital Common . . . . . .0 to +7V
V- to Digital Common . . . . . .0 to -16.5V
Analog Common to Digital Common. . . . . ±1V
Analog Input to Analog Common. . . . . ±15V
Control Inputs . . . . . 0 to V+
Digital Outputs (Blank Mode). . . . . 0 to V+
Power Dissipation. . . . . 800mW

**AD570 ORDERING GUIDE**

| Model | Package Number[1] | Temperature Range |
|---|---|---|
| AD570JN | 18-Pin Plastic DIP (N18A)[2] | 0 to +70°C |
| AD570JD | 18-Pin Ceramic DIP (D28A) | 0 to +70°C |
| AD570SD | 18-Pin Ceramic DIP (D18A) | -55°C to +125°C |
| AD570SD/883B | 18-Pin Ceramic DIP (D18A) | -55°C to +125°C |

[1] See Section 20 for package outline information.
[2] To be available June 1982.

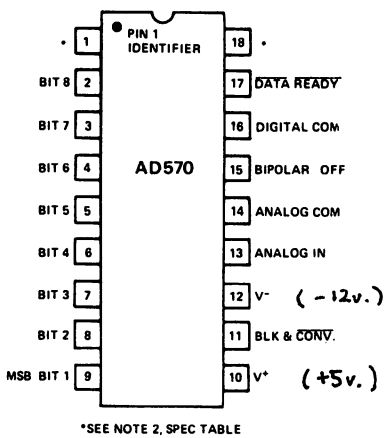



*Figure 1. AD570 Pin Connections*

## CONNECTING THE AD570 FOR STANDARD OPERATION

The AD570 contains all the active components required to perform a complete A/D conversion. Thus, for most situations, all that is necessary is connection of the power supply (+5 and -15), the analog input, and the conversion start pulse. But, there are some features and special connections which should be considered for achieving optimum performance. The functional pin-out is shown in Figure 1.

### FULL SCALE CALIBRATION

The 5kΩ thin film input resistor is laser trimmed to produce a current which matches the full scale current of the internal DAC—plus about 0.3%—when a full scale analog input voltage of 9.961 volts (10 volts − 1LSB) is applied at the input. The input resistor is trimmed in this way so that if a fine trimming potentiometer is inserted in series with the input signal, the input current at the full scale input voltage can be trimmed down to match the DAC full scale current as precisely as desired. However, for many applications the nominal 9.961 volt full scale can be achieved to sufficient accuracy by simply inserting a 15Ω resistor in series with the analog input to pin 14. Typical full scale calibration error will then be about ±2LSB or ±0.8%. If a more precise calibration is desired a 200Ω trimmer should be used instead. Set the analog input at 9.961 volts, and set the trimmer so that the output code is just at the transition between 11111110 and 11111111. Each LSB will then have a weight of 39.06mV. If a nominal full scale of 10.24 volts is desired (which makes the LSB have weight of exactly 40.00mV), a 50Ω resistor in series with a 200Ω trimmer (or a 500Ω trimmer with good resolution) should be used. Of course, larger full scale ranges can be arranged by using a larger input resistor, but linearity and full scale temperature coefficient may be compromised if the external resistor becomes a sizeable percentage of 5kΩ.

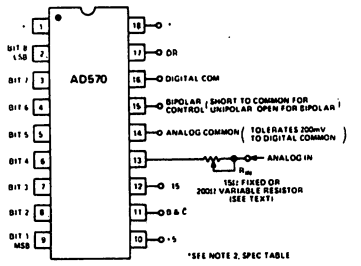



*Figure 2. Standard AD570 Connections*

### BIPOLAR OPERATION

The standard unipolar 0 to +10V range is obtained by shorting the bipolar offset control pin to digital common. If the pin is left open, the bipolar offset current will be switched into the comparator summing node, giving a -5V to +5V range with an offset binary output code. (-5.00 volts in will give a 8-bit code of 00000000; an input of 0.00 volts results in an output code of 10000000 and 4.96 volts at the input yields the 11111111 code.)

### ZERO OFFSET

The apparent zero point of the AD570 can be adjusted by inserting an offset voltage between the Analog Common of the device and the actual signal return or signal common. Figure 3 illustrates two methods of providing this offset. Figure 3A shows how the converter zero may be offset by up to ±3 bits to correct the device initial offset and/or input signal offsets. As shown, the circuit gives approximately symmetrical adjustment in unipolar mode. In bipolar mode R2 should be omitted to obtain a symmetrical range.

Figure 3B shows how to offset the zero code by 1/2LSB to provide a code transition between the nominal bit weights.

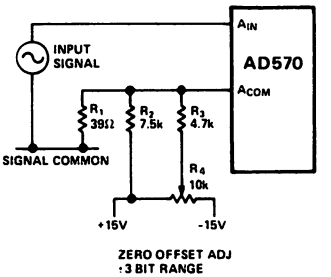



*Figure 3A.*

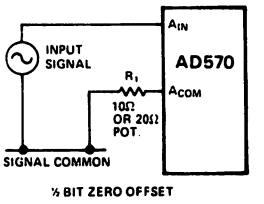



*Figure 3B.*

## CONTROL AND TIMING OF THE AD570

There are several important timing and control features on the AD570 which must be understood precisely to allow optimal interfacing to microprocessor or other types of control systems. All of these features are shown in the timing diagram in Figure 4.

The normal stand-by situation is shown at the left end of the drawing. The BLANK and $\overline{\text{CONVERT}}$ (B & $\overline{\text{C}}$) line is held high, the output lines will be "open", and the $\overline{\text{DATA READY}}$ ($\overline{\text{DR}}$) line will be high. This mode is the lowest power state

of the device (typically 150mW). When the (B & $\overline{C}$) line is brought low, the conversion cycle is initiated; but the $\overline{DR}$ and Data lines do not change state. When the conversion cycle is complete (typically 25$\mu$s), the $\overline{DR}$ line goes low, and within 500ns, the Data lines become active with the new data.

About 1.5$\mu$s after the B & $\overline{C}$ line is again brought high, the $\overline{DR}$ line will go high and the Data lines will go open. When the B & $\overline{C}$ line is again brought low, a new conversion will begin. The minimum pulse width for the B & $\overline{C}$ line to blank previous data and start a new conversion is 2$\mu$s. If the B & $\overline{C}$ line is brought high during a conversion, the conversion will stop, and the $\overline{DR}$ and Data lines will not change. If a 2$\mu$s or longer pulse is applied to the B & $\overline{C}$ line during a conversion, the converter will clear and start a new conversion cycle.

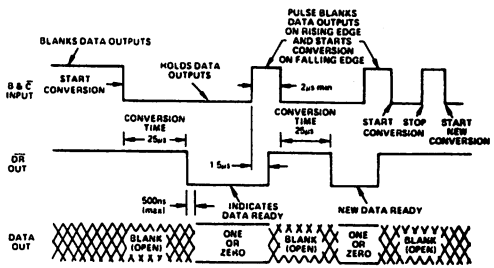
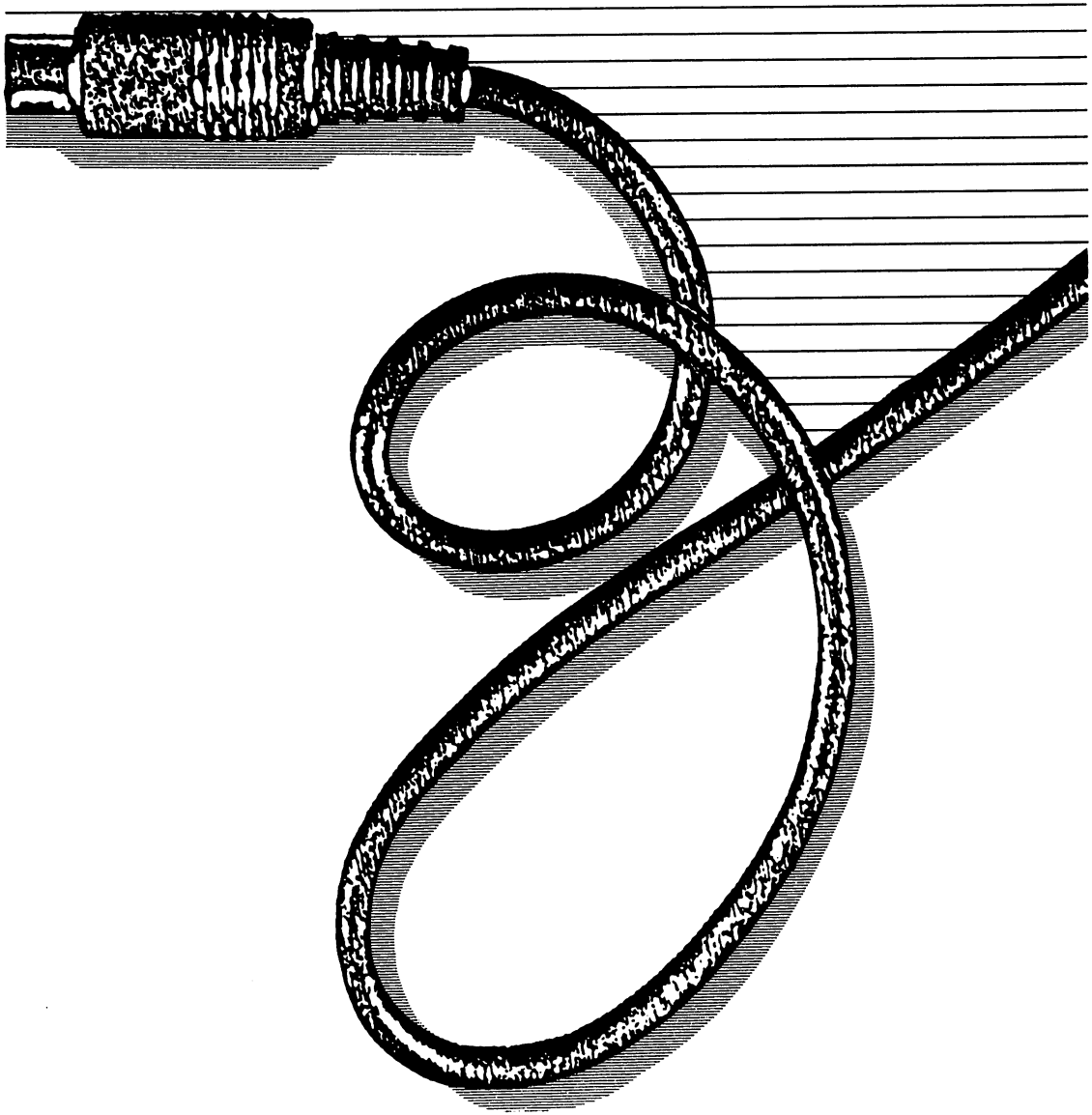
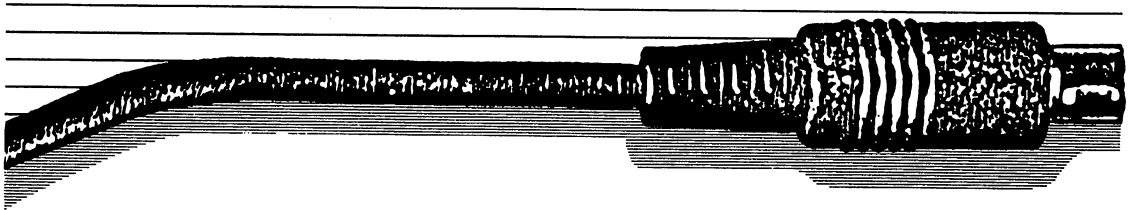



*Figure 4. AD570 Timing and Control Sequence*

# TIPS ON READING A SCHEMATIC DIAGRAM



# B



In this text we use a number of schematic diagrams to illustrate certain aspects of connecting the VIC-20 computer to the outside world. For those readers who are not familiar with electronic schematics, the following tips are given to help you understand what information is being presented.

To start, the symbols used in schematic diagrams represent physical devices. They are not meant to resemble the components they represent. Instead, they are standard, stylized symbols meant to be understood by convention as standing for their devices. The lines that interconnect the symbols represent physical wires. In Figure B.1, the symbol shown represents a transistor connected to a resistor. Each physical device has a a companion schematic symbol.

In digital logic many different symbols are used to represent the components of a circuit. We will discuss only those symbols that will aid you in reading the schematics presented in this text. However, if you understand these symbols it will be much easier to read schematics presented elsewhere. Let us discuss the schematic shown in Figure B.2 and explain all of its important points.

The first point to notice in Figure B.2 is that the schematic should be read from left to right, just like the words on a printed page. That is, digital information, in the form of electricity, flows from the components on the left to those on the right. This is true of most schematics. On the left side of Figure B.2 are the inputs to the circuit. These inputs are given a signal name so they may be identified wherever they are used in the schematic.

Each signal input in Figure B.2 also has a point of origin, indicating where the signal starts from. In this case, the inputs to the schematic will start from the edge connector pins of a VIC-20 I/O expansion slot. The edge connector pin number is shown in parentheses next to the signal name. For example, the data lines, D0–D7, originate from edge connector pin numbers 42–49, inclusive. The signals and pin numbers are listed in the VIC-20 documentation.

Following the signals D0–D7 farther to the right of the schematic, we see that they will connect to a rectangle. This rectangle represents a single integrated circuit. In this case the integrated circuit is a latch, labeled 74LS374. Each data line is shown connected to a specific

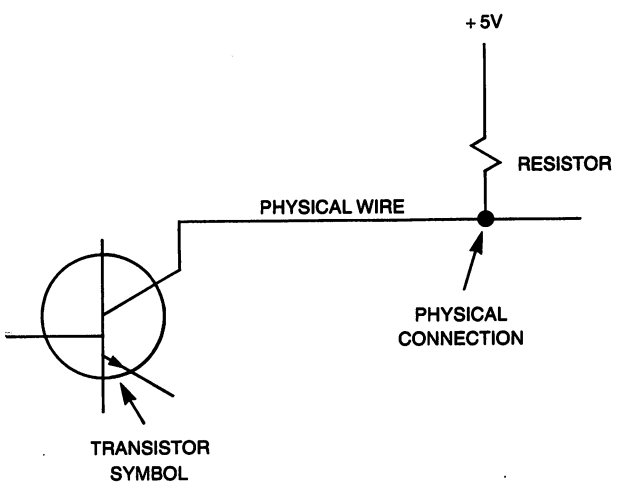



***Figure B.1**–Schematic diagram of a transistor connected to a resistor. Each element is identified.*

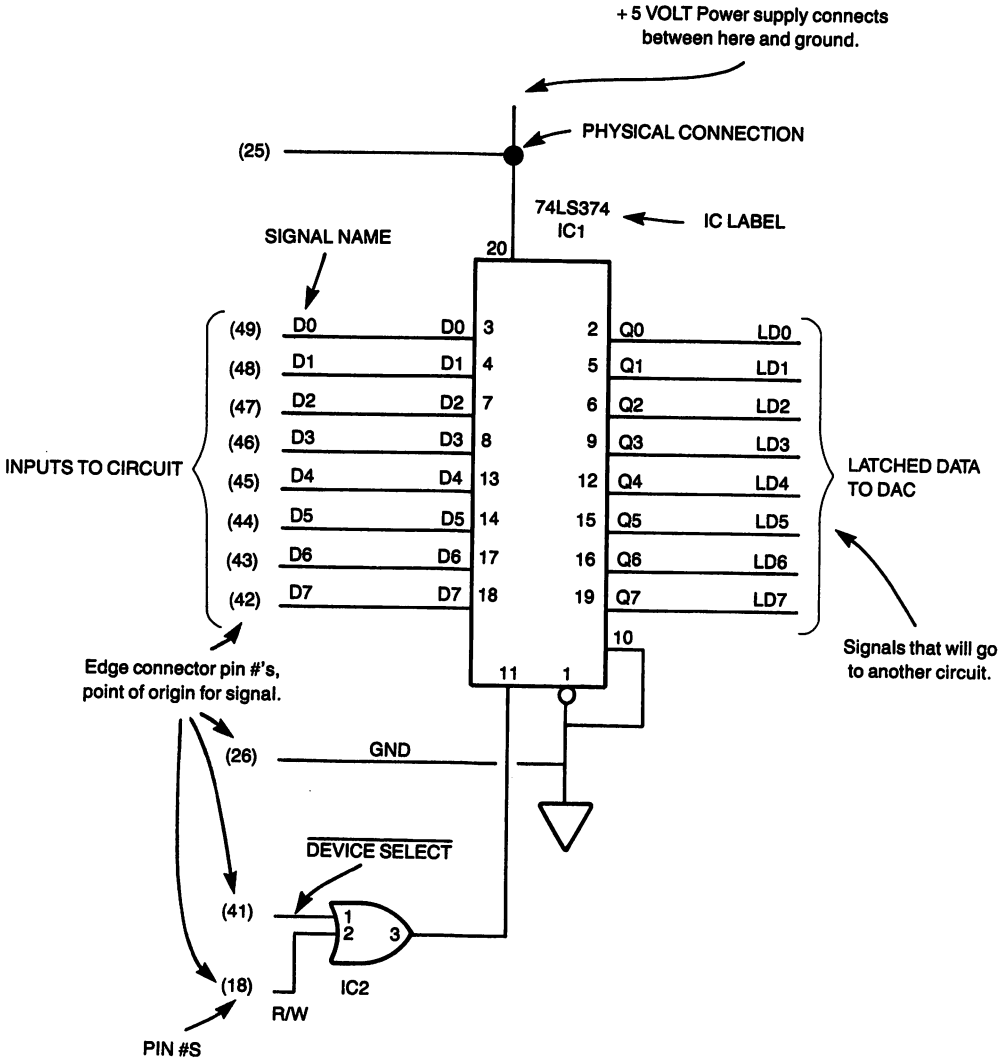



***Figure B.2**–A more complex schematic, with the elements identified.*

number on the rectangle. The specific number corresponds to the physical pin number of the integrated circuit. The pinout of this IC is shown in Figure B.3.

We see in this diagram that the integrated circuit has its pin numbers labeled in a "U" arrangement. Pin 1 will always be located at the upper left hand corner. Notice that Figure B.2 does not show the pin numbers in the same numerical order as Figure B.3. The pin numbers in Figure B.2 are obtained from Figure B.3, but the placement of the numbers in Figure B.2 does not represent numerical order; the numbers are arranged to allow the schematic to be drawn easily.

Let us now discuss another input line at the left of the schematic diagram B.2. This line is labeled $\overline{I/O2}$. Notice the heavy line over the signal name. This is an indication that this signal will perform its specified function (in this case, selecting the I/O slot) when it is in the logical 0 state. When the $\overline{I/O2}$ signal is in the logical 1 state, the I/O slot is not physically selected for electrical communication with the computer.

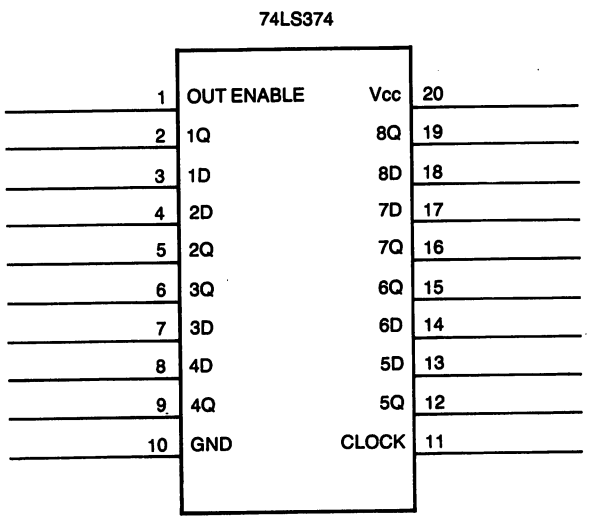



***Figure B.3**–The pins on this integrated circuit are shown in their correct numerical order. In Figure B.2, the layout was altered to simplify the drawing and make it more legible.*

The $\overline{I/O2}$ select line originates from edge connector pin T. It is then connected to the symbol for a logical OR gate, shown in Figure B.2. Pin 1 of the OR gate represents the physical pin number of the integrated circuit package.

A single integrated circuit package may contain up to four logical OR gates as shown in Figure B.4.

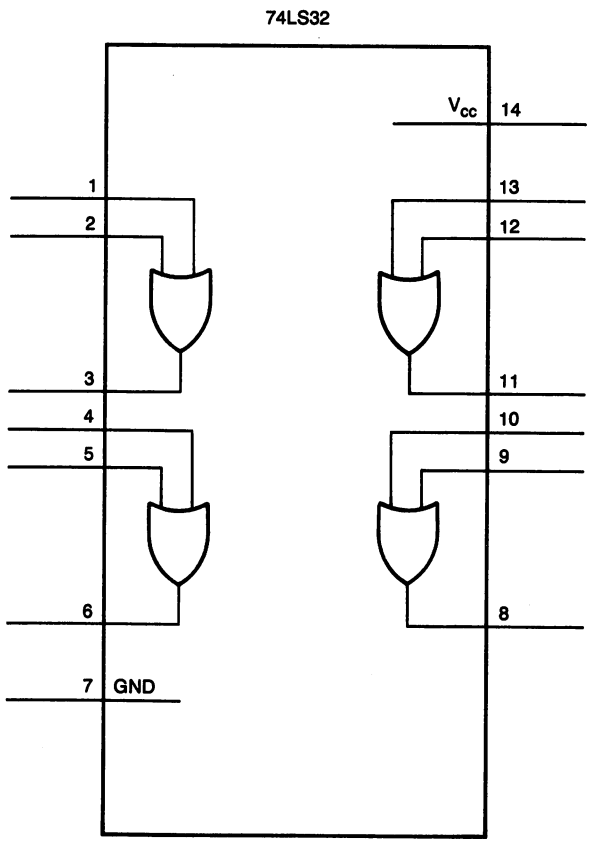



***Figure B.4**–All four OR gates are located in the same device package.*

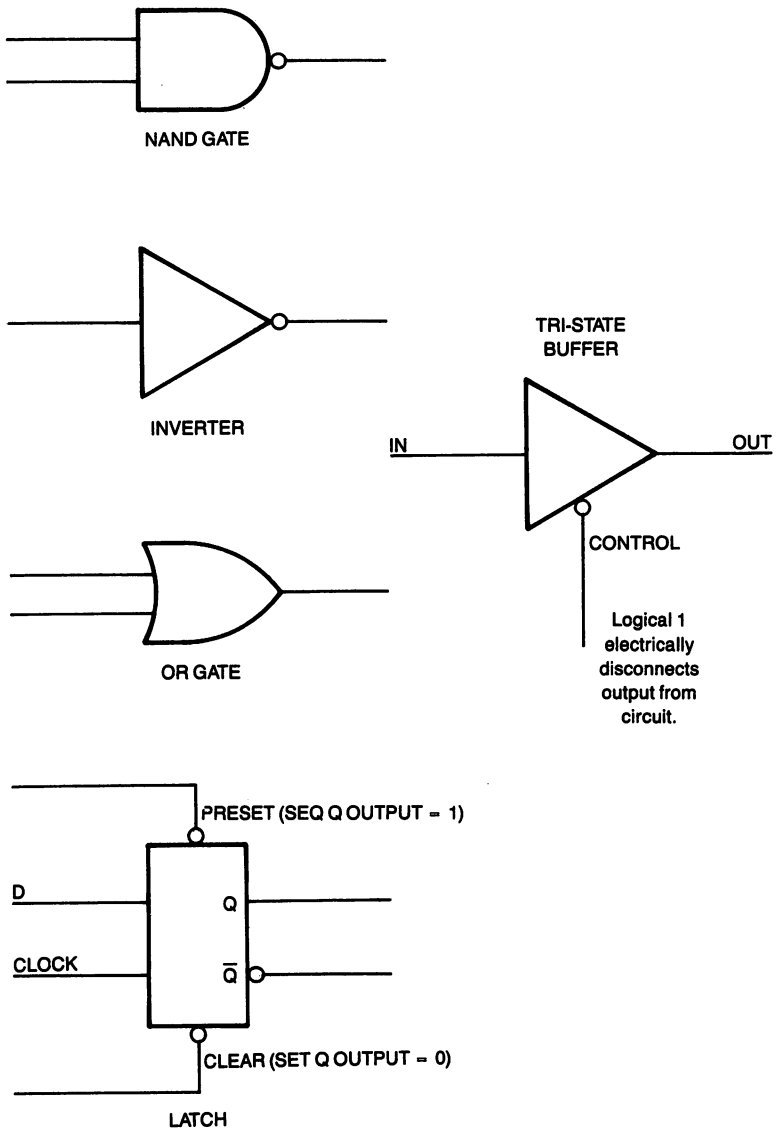


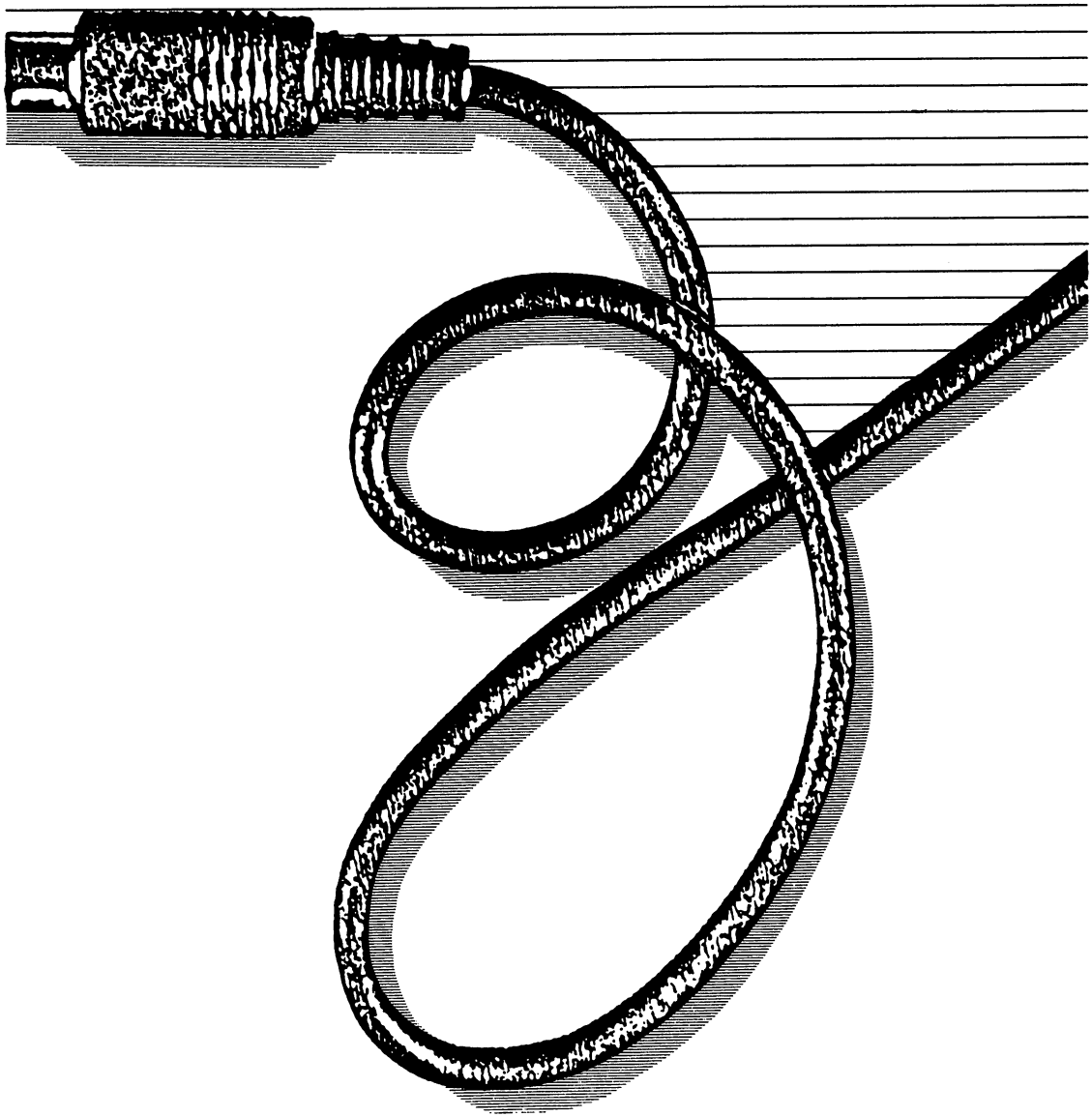

***Figure B.5**–The logic symbols used in this book.*

In Figure B.2, the output pin 3 of the OR gate is connected to pin 4 of another OR gate. Both of these OR gates are contained in the same integrated-circuit package. We know this because both of the OR gates are labeled IC2. The "IC" stands for Integrated Circuit. This method of labeling is often used to denote gates that reside in the same physical package on the circuit board. Many schematics in this book show several components with the same IC number.

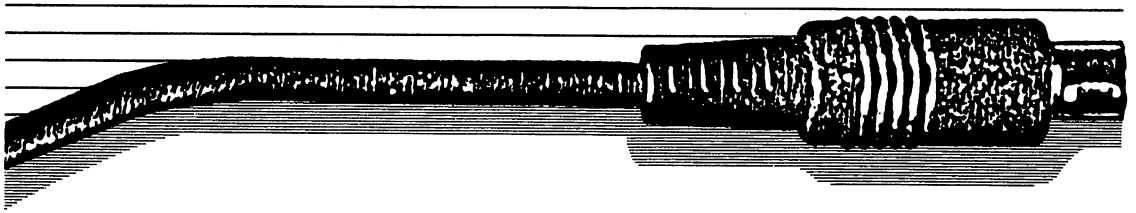
Output pin 6 of the OR gate is connected to input pin 11 of the 74LS374 integrated circuit we discussed previously. The outputs of the 74LS374 are not connected to anything in Figure B.2. If they were, the lines would be connected to some other symbol in the drawing. Instead, they lead off the page, sending latched data to, say, a DAC. Notice that the flow of the schematic is from left to right, input to output, as stated before.

Using the information presented here, you should be able to read and comprehend the schematic diagrams shown in this text. All of the special symbols used, like the symbol for a light-emitting diode, are discussed in the section of the text where they are used. This text uses only a few of the many logic symbols available. The ones used are shown in Figure B.5. The pinouts for each of the physical integrated circuits used in this text are given in the data sheets contained in Appendix A.

# GLOSSARY OF SELECTED TERMS



# C



**AC Appliance** AC is an abbreviation for Alternating Current. This description is given to any home appliance that can be plugged into a wall socket. A toaster, a desk lamp, an electric can opener, and a coffee maker are all examples of AC appliances.

**Analog event** An event in nature that can have any value for its output. Some common analog events are temperature, pressure and brightness.

**Analog-to-Digital Conversion (ADC)** An electrical process by which an analog voltage is converted into its digital equivalent, so it can be input to a home computer (or any type of digital computer).

**Analog voltage** An analog voltage is a voltage output from any source that can take on any numeric value. For example, 15.2345 V is an analog voltage.

**BASIC** (Beginner's All-purpose Symbolic Instruction Code) A computer programming language used by the VIC-20 computer and most other home computers.

**Binary** A description given to a set of values that may have two, and only two, possible outcomes. For example, the binary number system uses only two digits, 1 and 0.

**Bit** A single binary digit in a computer word. A bit may have the values 1 or 0.

**Black box** This colloquial term is used to describe any electronic circuit that performs a certain function, but whose internal operation need not be understood by the user, and usually isn't. A black box is inherently mysterious to its user.

**Board** An abbreviation for the term *circuit board.*

**Byte** A group of eight bits. An example of a byte would be 00101100. For an 8-bit computer like the VIC-20, *byte* and (data) *word* are synonymous.

**Card** Another way of saying *board.*

**Circuit** A collection of electronic components wired together to perform a certain function, or the electrical path between the components. A circuit may consist of resistors, capacitors, transistors, digital IC's, or any other electronic elements.

**Computer control** The operation of a device or system under the direction of a computer.

**Data** Information output from or input to the computer. It takes the physical form of electrical pulses at one of two possible voltage levels.

**Digital-to-Analog Conversion (DAC)** An electrical process by which an analog output voltage is produced by a specific combination of binary inputs to a piece of hardware called a digital-to-analog converter.

**DIP** An abbreviation for *Dual In-line Package,* an electronic package used to mount integrated circuits. It is rectangular and has leads (or "pins") extending from both of its long sides in a symmetrical pattern.

**Flip-flop** A group of digital electronic components that have the characteristic of storing information. Flip-flops are *bistable;* that is, they have two stable states, 1 and 0.

**Flowchart** A visual representation of the logical paths a computer program will follow as the instructions are executed.

**4.7 KΩ** An abbreviation for 4700 ohms, a resistance used in some circuits in this book. The letter K is the metric symbol for 1000. Omega is the symbol for ohm, the unit of resistance.

**Ground** The point in a system that has a voltage potential of 0.0 volts.

**Ground potential** A voltage potential of approximately 0.0 volts. *Potential* is the difference in voltage between two points.

**I/O** Abbreviation for Input/Output.

**I/O address** A logical memory address that is assigned to a specific peripheral device.

**$\overline{\text{I/O2}}$ line** A single logical line that will become active when the expansion connector on the VIC-20 computer is selected with the software.

**Input** A signal sent to a computer or other device, or the act of sending a signal. During an I/O operation the computer can input, or take in, data.

**Integrated Circuit (IC)** An electronic component consisting of one or more circuit elements fabricated on a single chip of silicon, and usually packaged in a DIP.

**Latch** An electronic device capable of storing a single bit of binary information when electrically instructed to do so. An 8-bit latch will store eight bits of binary information at the same time.

**Light-Emitting Diode (LED)** An electronic device that has the physical property of emitting a certain wavelength of light when current is passed through it in the forward direction.

**Logical 0** One of the two possible binary states that a digital logic circuit can reside in. For the Vic-20 computer, a logical 0 is equivalent to a voltage level of less than or equal to .8 volts.

**Logical 1** The second of the two possible binary states that a digital logic circuit can reside in. In the Vic-20 computer, a logical 1 is equivalent to a voltage output greater than 2.0 volts.

**Logic gate** A digital hardware element that will perform one of the Boolean logical functions, such as AND or OR.

**Nibble** A group of four bits. 0011 is an example of a nibble. A byte consists of two nibbles.

**Output** Information sent from a computer to another device, or the action of sending it. When the computer is sending data to a peripheral device for control, it is performing output. Broadly, an electrical quantity produced by a circuit or the physical action that results. The output of a lamp is light; the output of a loudspeaker is sound.

**Output port** Any output address that the computer can send data to.

**PEEK** An instruction in BASIC that will allow data to be input from a valid memory address.

**PEEK address** The address used in a PEEK instruction.

**POKE** An instruction in BASIC that allows data to be output to a valid memory address.

**POKE address** The memory address that is specified during the execution of the POKE instruction.

**Resistance** An electrical property that impedes the flow of electrons.

**Resistor** A physical element that has the electrical property of resistance. It is usually tubular, with leads extending from both ends.

**Schematic diagram** A drawing using electronic symbols to represent the component interconnections of a circuit.

**74LS__** A numeric label given to a family of integrated circuits used in this book. LS is an abbreviation for Low-power Schottky, a type of design technology. The blank in the label can be any 2- or 3-digit number. This number makes it possible to look up the integrated circuit in a data book.

**Solid-State Relay (SSR)** An electronic device that will open and close two electronic contacts without any mechanical parts.

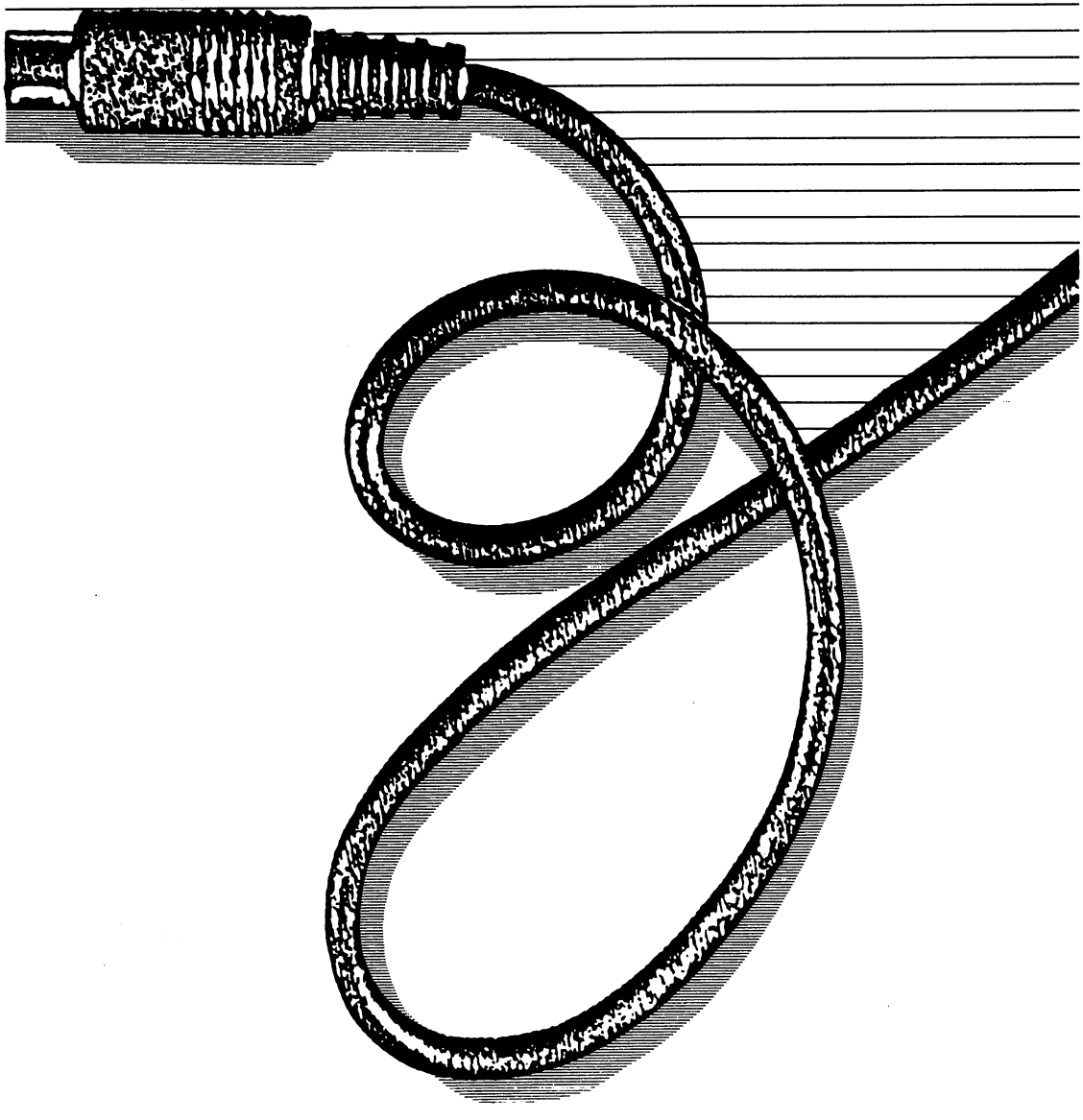
**Transducer** An electronic device that will change a physical action into an electrical equivalent, or vice-versa. For example, a temperature transducer will change the temperature it senses into an equivalent electrical value.

**Transistor** An electronic component that can be made to amplify electronic signals.

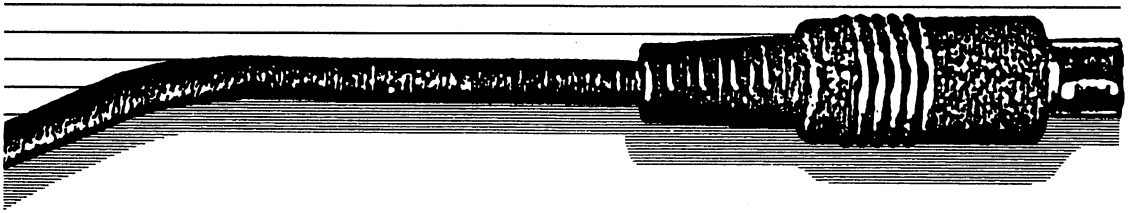
**Transistor-Transistor Logic (TTL)** The type of logic family used in the circuits of the VIC-20 computer.

**VR/$\overline{W}$ line** A signal line that is used to electrically inform the I/O circuits whether the computer is reading or writing during this memory cycle.

# LIST OF VENDORS



# D



Information concerning the prices and availability of various hardware devices described in this book can be obtained by writing to the outlets listed below. The list is not meant to be exhaustive.

TTL components (logic gates, inverters, latches, etc.):

Jameco Electronics
1355 Shoreway Rd.
Belmont, CA 94002

Quest Electronics
PO Box 4430
Santa Clara, CA 95054

Analog-to-digital and digital-to-analog converters, along with several different peripheral devices for microcomputer I/O:

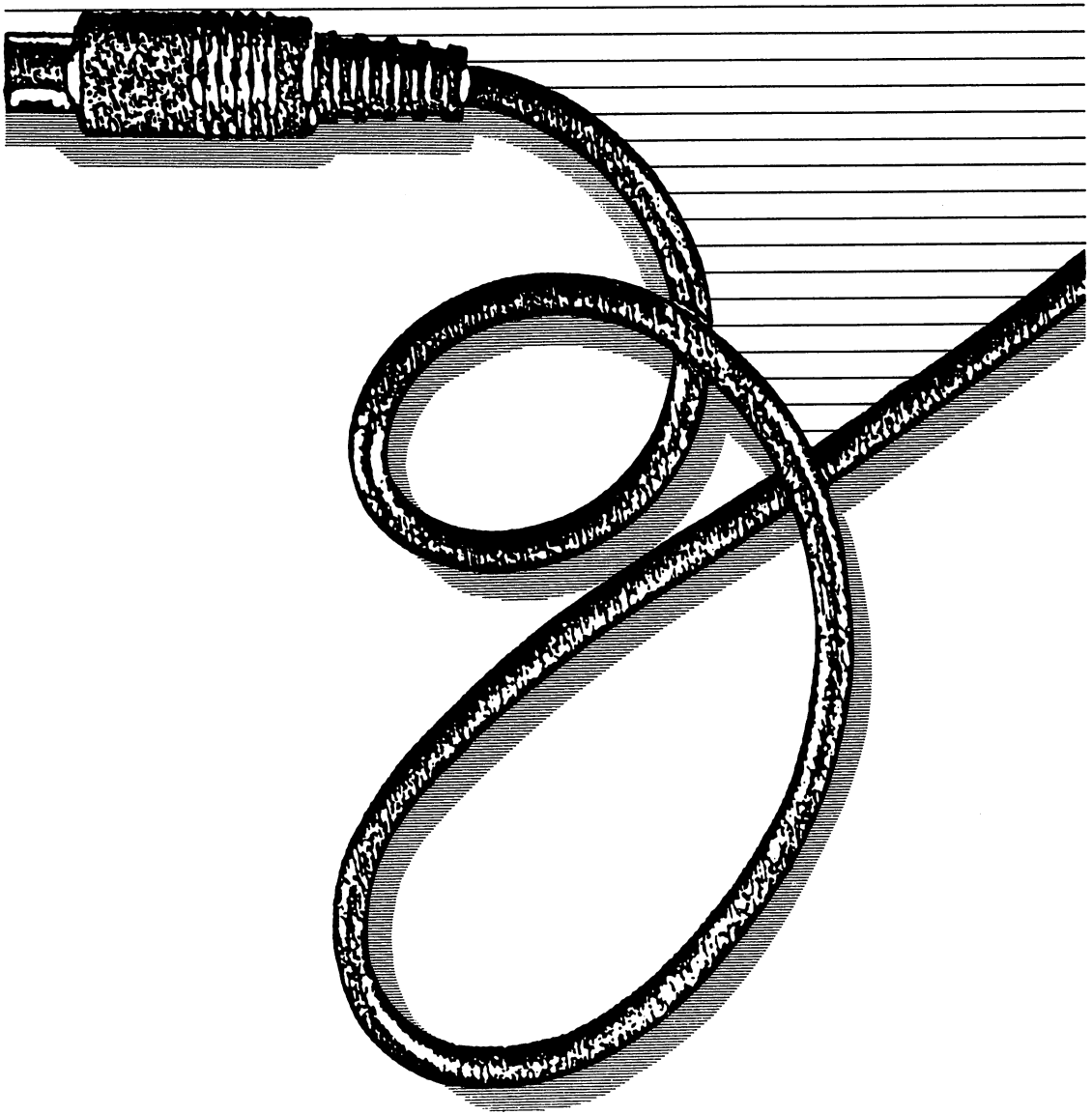
Microprocessor Training Inc.
14 East Eighth St.
New York, N.Y. 10003

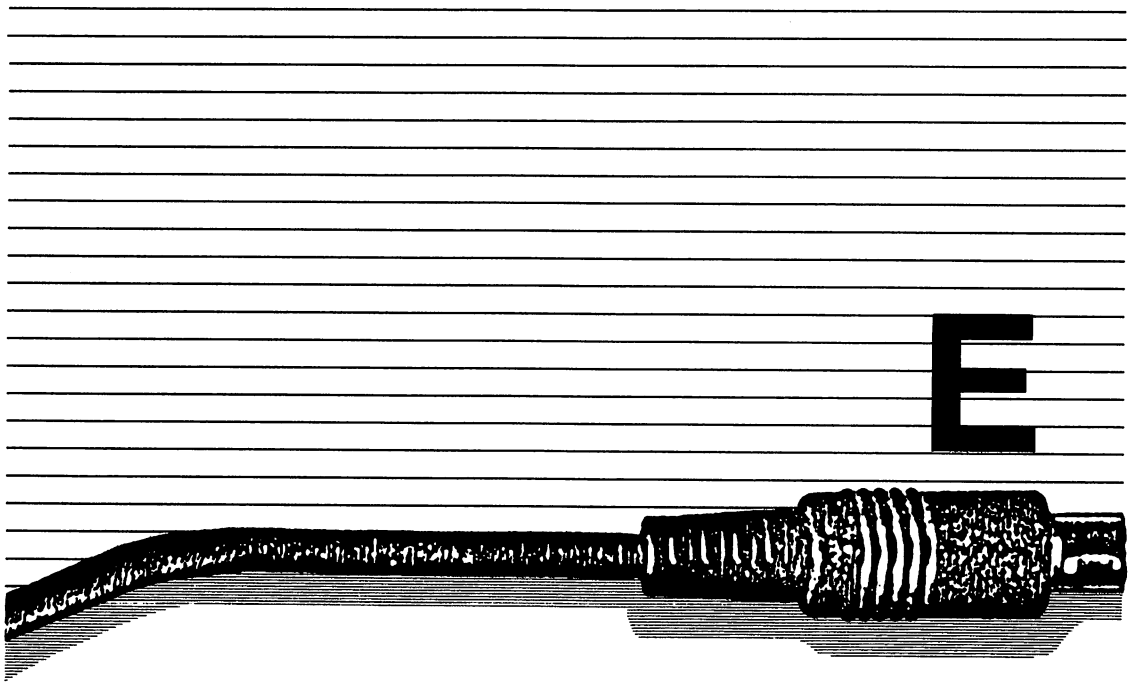
CMS I/O board for the VIC-20 computer:

Creative Microprocessor Systems, Inc.
PO Box 1538
Los Gatos, CA 95030

408-354-2011
(30 days)

# THE VOTRAX PHONETIC SPEECH DICTIONARY





## GENERAL DESCRIPTION OF THE SC-01 CHIP

The SC-01 Speech Synthesizer is a completely self-contained solid-state device. This single chip phonetically synthesizes continuous speech of unlimited vocabulary from low data-rate inputs.

Speech is synthetized by combining phonemes (the building blocks of speech) in the appropriate sequence. The SC-01 Speech Synthesizer contains 64 different phonemes which are accessed by a 6-bit code. It is the proper sequential combination on these phoneme codes that creates continuous speech.

## PHONEME DESCRIPTION

**Figure 6.1** lists the 64 phonemes produced by the SC-01. Each sound is represented by its Votrax® phoneme code and is accompanied by its phoneme symbol and an example. The underlined segment of the example word demonstrates the phoneme use, i.e., the sound to be pronounced.

**Table 1** provides the phoneme sequences used to produce vowels in the group called diphthongs, (2 vowel sounds in sequence, identified as a single sound, e.g., the long "i" vowel).

**Table 2** lists approximately 1400 words and the sequences of phonemes that produce them.

### TABLE 1: DIPHTHONG CHART

| Phoneme Combination | Key Words |
|---|---|
| A1-AY-Y | fate, maid |
| AH1-EH3-Y | find, wide |
| UH3-AH2-Y | fight, white |
| AH1-I3-UH3-L | file, smile |
| 01-UH3-Y | foy, boy |
| 01-I3-UH3-L | foil, spoil |
| AH1-02-U1 | found, cow |
| UH3-AH2-U1 | foust, house |
| O1-U1 | float, note |
| Y1-IU-U1 | few, you, music |
| AY-I1 | fear, beer |

| Word | Phonetic Program |
|---|---|
| A | A1, AY, Y |
| a-2 | UH2, UH3 |
| able | A1, Y, B, UH3, L |
| abort | UH1, B, O2, O2, R, T |
| about | UH1, B, UH2, AH2, U1, T |
| above | UH1, B, UH1, UH3, V |
| accept | EH1, K, PAO, S, EH1, EH3, P, T |
| access | AE1, EH3, K, PAO, S, EH1, EH3, S |
| account | UH1, K, AH1, UH3, W, N, T |
| acid | AE1, EH3, S, I1 D |
| act | AE1, EH3, K, T |
| active | AE1, EH3, K, T, I1, V |
| actual | AE1, EH3, K, T, CH, U1, UH3, L |
| add | AE1, EH3, D |
| address | AE1, EH3, D, R, EH1, EH3, S |
| ade | (use "aid" program) |
| adjust | UH1, D, J, UH1, UH3, S, T |
| adjucent | UH1, D, J, A1, AY, S, EH3, N, T |
| advance | AE1, EH3, D, V, AE1, EH3, N, T, S |
| advise | AE1, EH3, D, V, AH1, EH3, Y, Z |
| affect | UH1, F, EH1, EH3, K, T |
| after | AE1, EH3, F, T, ER |
| again | UH1, G, A2, EH1, N |
| age | A1, AY, Y, D, J |
| agent | A1, Y, D, J, EH3, N, T |
| ahead | UH1, H, EH1, EH3, D |
| aid | A1, AY, Y, D |
| air | EH2, EH2, R |
| alarm | UH1, L, AH1, R, M |
| alert | UH1, L, ER, R, T |
| all | AW, L |
| allocate | AE1, UH3, L, UH2, K, A1, Y, T |
| allow | UH1, L, AH1, UH3, U1 |
| alpha | AE1, AW2, L, F, UH1 |
| already | AW, L, R, EH1, EH3, D, Y |
| also | AW, L, S, O1, U1 |
| altitude | AE1, UH3, L, T, I2, T, IU, U1, U1, D |
| aluminum | UH1, L, IU, U1, M, I3, N, UH1, M |
| am | AE1, EH3, M |
| america | UH1, M, EH1, R, I3, K, UH2, UH3 |
| amount | UH1, M, AH1, UH3, W, N, T |
| amp | AE1, EH3, M, P |
| amplify | AE1, EH3, M, P, L, I3, F, AH1, EH3, AY |
| an | AE1, EH3, N |
| and | AE1, EH3, N, D |

| Word | Phonetic Program |
|---|---|
| angle | AE1, EH3, NG, G, UH3, L |
| another | UH1, N, UH1, UH3, THV, ER |
| answer | AE1, EH3, N, S, ER |
| any | EH2, EH2, N, Y |
| apostrophe | UH1, P, AH1, UH3, S, T, R, UH3, F, Y |
| approach | UH1, P, R, O1, U1, T, CH |
| approve | UH1, P, R, IU, U1, U1, V |
| approximate | UH1, P, R, AH1, K, PAO, S, EH3, M, I3, T |
| approximate-2 | UH1, P, R, AH1, K, PAO, S, EH3, M, A2, Y, T |
| april | A1, Y, P, R, UH2, L |
| architect | AH1, R, K, UH2, T, EH3, EH2, K, T |
| are | (see "R" program) |
| area | EH1, EH3, R, Y, UH1 |
| arrive | UH1, R, AH1, EH3, Y, V |
| arrow | EH1, EH3, R, O1, U1 |
| article | AH1, R, T, EH3, K, UH3, L |
| as | AE1, EH3, Z |
| ASCII | AE1, EH3, S, K, Y |
| ask | AE1, EH3, S, K |
| assemble | UH1, S, EH1, EH3, M, B, UH3, L |
| asset | AE1, EH3, S, EH1, T |
| assign | UH1, S, AH1, EH3, Y, N |
| assist | UH1, S, I1, I3, S, T |
| associate | UH1, S, O1, SH, Y, A1, Y, T |
| associate-2 | UH1, S, O1, SH, Y, I2, T |
| assume | UH1, S, IU, U1, M |
| at | AE1, EH3, T |
| ate | (see "eight" program) |
| attach | UH1, T, AE1, EH3, T, CH |
| attempt | UH1, T, EH1, EH3, M, P, T |
| attend | UH1, T, EH1, EH3, N, D |
| audio | AW, D, Y, O1, U1 |
| august | AW2, AW2, G, EH2, S, T |
| authorize | AW2, AW2, TH, ER, AH1, Y, Z |
| automatic | AW2, AW2, DT, UH3, M, AE1, EH3, DT, I3, K |
| available | UH1, V, A1, Y, L, UH3, B, UH3, L |
| average | AE1, EH3, V, R, I1, D, J |
| avoid | UH1, V, O1, UH3, I3, AY, D |
| B | B, E1, Y |
| back | B, AE1, AE1, K |
| bad | B, AE1, AE1, D |
| badge | B, AE1, AE1, D, J |
| bag | B, AE1, AE1, G |

| Word | Phonetic Program |
|---|---|
| balance | B, AE1, AH2, L, I3, N, DT, S |
| ball | B, AW2, AW1, L |
| band | B, AE1, EH3, N, D |
| bank | B, AE1, I3, NG, K |
| bar | B, AH1, UH3, R |
| base | B, A1, AY, Y, S |
| basic | B, A1, Y, S, I2, K |
| bat | B, AE1, EH3, T |
| batch | B, AE1, EH3, T, CH |
| bath | B, AE1, AE1, EH3, TH |
| battery | B, AE1, EH3, T, ER, Y |
| be | (use "B" program) |
| bed | B, EH1, EH3, D |
| been | B, EH1, EH3, N |
| beep | B, E1, Y, P |
| before | B, Y, F, O2, O2, R |
| begin | B, Y, G, I1, I3, N |
| bell | B, EH1, UH3, L |
| below | B, Y, L, UH3, O2, U1 |
| bend | B, EH1, EH3, N, D |
| best | B, EH1, EH3, S, T |
| beta | B, A2, A2, AY, T, UH2 |
| better | B, EH1, EH3, T, ER |
| between | B, Y, T, W, E1, Y, N |
| bid | B, I1, I3, D |
| big | B, I1, I3, G |
| bill | B, I1, I3, L |
| billion | B, I1, I3, L, Y, UH3, N |
| bin | B, I1, I3, N |
| binary | B, AH1, Y, N, EH3, EH3, ER, Y |
| birthday | B, ER, R, TH, D, A1, I3, Y |
| bit | B, I1, I3, T |
| bite | B, UH3, AH2, Y, T |
| black | B, L, AE1, EH3, K |
| blank | B, L, AE1, EH3, NG, K |
| blew | (use "blue" program) |
| blind | B, L, AH1, EH3, Y, N, D |
| block | B, L, AH1, UH3, K |
| blown | B, L, O1, U1, N |
| blue | B, L, IU, U1, U1 |
| blur | B, L, ER, R |
| board | B, O1, O2, R, D |
| bolt | B, O2, O2, L, T |
| bond | B, AH1, UH3, N, D |
| book | B, OO1, OO1, K |
| bored | (use "board" program) |
| boss | B, AW1, AW2, S |
| bother | B, AH1, UH3, THV, ER |
| bottom | B, AH1, UH3, T, UH1, M |
| bought | B, AW1, AW2, T |
| box | B, AH1, UH3, K, PAO, S |
| brace | B, R, A1, Y, S |
| brain | B, R, A1, Y, N |
| brake | B, R, A1, Y, K |
| branch | B, R, AE1, EH3, N, T, CH |
| bravo | B, R, AH1, UH3, V, O1, U1 |
| break | (use "brake" program) |
| bridge | B, R, I1, I3, D, J |
| brief | B, R, AY, Y, F |
| bright | B, R, UH3, AH2, Y, T |
| bring | B, R, I1, I3, NG |
| broke | B, R, O1, U1, K |
| brought | B, R, AW, T |
| brown | B, R, AH1, UH3, U1, N |
| bubble | B, UH1, UH2, B, UH3, L |
| budget | B, UH1, UH3, D, J, I2, T |
| bug | B, UH1, UH2, G |
| build | B, I2, I2, L, D |
| bus | B, UH1, UH2, S |
| business | B, I3, I3, Z, N, EH2, S |
| busy | B, I3, I2, Z, Y |
| but | B, UH1, UH2, T |
| button | B, UH1, UH3, T, EH3, N |
| buy | B, AH1, EH3, I3, Y |
| by | B, AH1, EH3, I3, Y |
| bye | B, AH1, EH3, I3, Y |
| byte | (use "bite" program) |
| C | S, E1, Y |
| cable | K, A1, Y, B, UH3, L |
| calendar | K, AE1, UH3, L, I3, N, D, ER |
| calibrate | K, AE1, UH3, L, UH3, B, R, A1, Y, T |
| call | K, AW2, AW1, L |
| came | K, A1, AY, Y, M |
| can | K, AE1, EH3, N |
| cancel | K, AE1, EH3, N, S, UH3, L |
| capable | K, A1, Y, P, UH3, B, UH3, L |
| capacitor | K, UH2, P, AE1, EH3, S, EH3, T, ER |
| capacity | K, UH2, P, AE1, EH3, S, I3, DT, Y |
| car | K, AH1, UH3, R |
| card | K, AH1, R, D |
| care | K, EH3, EH3, ER |
| carpenter | K, AH1, R, P, I3, N, D, ER |
| carriage | K, EH2, EH3, R, I1, D, J |
| carry | K, EH2, EH3, R, Y |
| carton | K, AH1, R, T, I3, N |
| case | K, A1, AY, Y, S |

| Word | Phonetic Program |
|---|---|
| cash | K, AE1, EH3, SH |
| cassette | K, UH1, S, EH1, EH3, T |
| cassette-2 | K, A2, AY, S, EH1, EH2, T |
| category | K, AE1, EH3, DT, UH3, G, O1, R, Y |
| catalog | K, AE1, EH3, DT, UH3, L, AW2, AW2, G |
| caution | K, AW2, AW1, SH, UH3, N |
| cent | S, EH1, EH3, N, T |
| center | S, EH1, EH3, N, T, ER |
| centi | S, EH1. EH3, N, T, I1, I3 |
| centigrade | S, EH1, N, T, I3, G, R, A1, Y, D |
| certify | S, R, R, T, I3, F, AH1, Y |
| change | T, CH, A1, AY, Y, N, D, J |
| character | K, EH1, R, EH1, K, T, ER |
| charge | T, CH, AH1, R, D, J |
| charlie | T, CH, AH1, R, L, Y |
| chart | T, CH, AH1, R, T |
| check | T, CH, EH1, EH3, K |
| cheer | T, CH, AY, I2, R |
| chip | T, CH, I1, I3, P |
| choice | T, CH, O1, UH3, I3, AY, S |
| circle | S, ER, R, K, UH3, L |
| circuit | S, R, R, K, I2, T |
| city | S, I1, T, Y |
| claim | K, L, A1, AY, Y, M |
| class | K, L, AE1, EH3, S |
| clean | K, L, E1, AY, N |
| clear | K, L, AY, I3, R |
| clerk | K, L, ER, K |
| clip | K, L, I1, I3, P |
| clock | K, L, AH1, UH3, K |
| close | K, L, UH3, O1, U1, Z |
| close-2 | K, L, UH3, O2, U1, S |
| cloud | K, L, AH1, UH3, W, D |
| coarse | K, O1, O2, R, S |
| code | K, OO1, O2, U1, D |
| coin | K, O1, UH3, I3, AY, N |
| collar | K, AH1, UH3, L, ER |
| collect | K, UH1, L, EH1, K, T |
| colon | K, OO1, O2, U1, L, I2, N |
| color | K, UH2, UH2, L, ER |
| column | K, AH1, UH3, L, UH3, M |
| combine | K, UH2, M, B, AH1, EH3, Y, N |
| comma | K, AH1, UH3, M, UH1 |
| command | K, UH2, M, AE, EH3, N, D |
| commerce | K, AH1, UH3, M, ER, S |
| commercial | K, UH1, UH3, M, ER, SH, UH3, L |
| communicate | K, UH2, M, Y1, IU, U1, N, I3, K, A1, Y, T |
| company | K, UH1, UH3, M, P, EH3, N, Y |
| compare | K, UH1, UH3, M, P, EH3, EH3, ER |
| compile | K, UH1, UH3, M, P, AH1, EH3, I3, UH3, L |
| complete | K, UH1, UH3, M, P, L, AY, Y, T |
| comply | K, UH1, UH3, M, P, L, AH1, EH3, Y |
| component | K, UH2, M, P, O2, O1, N, EH2, N, T |
| computer | K, UH1, M, P, Y1, IU, U1, T, ER |
| conceal | K, UH1, N, S, E1, AY, L |
| condense | K, UH1, N, D, EH1, EH3, N, S |
| condition | K, UH1, N, D, I1, I3, SH, UH3, N |
| confirm | K, UH1, N, F, ER, R, M |
| confuse | K, UH1, N, F, Y1, IU, U1, U1, Z |
| confusion | K, UH1, N, F, Y1, IU, U1, U1, ZH, UH3, N |
| congratulations | K, UH1, N, G, R, AE1, D, J, UH3, L, A1, AY, SH, UH3, N, Z |
| connect | K, UH1, N, EH1, EH3, K, T |
| console | K, AH1, UH3, N, S, O1, U1, L |
| console-2 | K, UH1, N, S, O1, O2, L |
| consult | K, UH1, N, S, UH1, UH2, L, T |
| consume | K, UH1, N, S, IU, U1, U1, M |
| contain | K, UH3, UH3, N, T, A1, AY, Y, N |
| continue | K, UH1, N, T, I1, I3, N, Y1, IU, U1 |
| contract | K, AH1, UH3, N, T, R, AE1, EH3, K, T |
| contrast | K, AH1, UH3, N, T, R, AE1, EH3, S, T |
| control | K, UH1, N, T, R, O1, O2, L |
| convenient | K, UH2, N, V, E1, N, AY, EH3, N, T |
| copper | K, AH1, UH3, P, ER |
| copy | K, AH1, UH3, P, Y |
| correct | K, O2, O2, R, EH1, EH3, K, T |
| correspond | K, O1, R, I3, S, P, AH1, AH2, N, D |
| cosine | K, O1, U1, S, AH1, Y, N |
| cost | K, AW2, AW1, S, T |
| could | K, IU, IU, OO1, D |
| count | K, AH1, UH3, W, N, T |
| country | K, UH1, N, T, R, Y |
| couple | K, UH3, UH1, P, UH3, L |
| courage | K, ER, R, I3, D, J |
| course | K, O1, O2, R, S |
| court | K, O1, O2, R, T |
| cover | K, UH1, UH3, V, ER |
| crane | K, R, A1, AY, Y, N |

| Word | Phonetic Program |
|---|---|
| crash | K, R, AE1, EH3, SH |
| crease | K, R, E1, Y, S |
| create | K, R, Y, A1, Y, T |
| creation | K, R, Y, A1, Y, SH, UH3, N |
| credit | K, R, EH1, EH3, D, I1, T |
| crew | K, R, IU, U1, U1 |
| critical | K, R, I1, T, I3, K, UH3, L |
| cross | K, R, AW, S |
| crowd | K, R, AH1, UH3, U1, D |
| cry | K, R, AH1, EH3, I3, Y |
| cue | (use "Q" program) |
| cup | K, UH1, UH2, P |
| curious | K, Y, ER, Y, UH1, S |
| current | K, ER, R, EH3, N, T |
| currency | K, ER, R, I2, N, DT, S, Y |
| curse | K, ER, R, S |
| curve | K, ER, R, V |
| customer | K, UH1, UH2, S, T, UH1, M, ER |
| cut | K, UH1, UH2, T |
| cycle | S, UH3, AH2, Y, K, UH3, L |
| D | D, E1, Y |
| daily | D, A1, AY, Y, L, Y |
| damage | D, AE1, EH3, M, I1, D, J |
| danger | D, A1, AY, Y, N, D, J, ER |
| dark | D, AH1, R, K |
| dash | D, AE1, EH3, SH |
| data | D, A1, Y, DT, UH1 |
| date | D, A1, AY, Y, T |
| day | D, A1, I3, Y |
| dead | D, EH1, EH3, F |
| dealer | D, E1, AY, L, ER |
| dear | D, AY, I3, R |
| debit | D, EH1, EH3, B, I2, T |
| debt | D, EH1, EH3, T |
| december | D, Y, S, EH1, EH3, M, B, ER |
| decide | D, Y, S, AH1, EH3, Y, D |
| decimal | D, EH1, S, M, UH3, L |
| decision | D, Y, S, I1, ZH, UH3, N |
| decline | D, Y, K, L, AH1, EH3, Y, N |
| decrease | D, Y, K, R, E1, Y, S |
| deduct | D, Y, D, UH1, UH2, K, T |
| deep | D, E1, Y, P |
| deer | (use "dear" program) |
| defeat | D, Y, F, E1, AY, T |
| defend | D, Y, F, EH1, EH3, N, D |
| defensive | D, Y, F, EH1, EH3, N, S, I1, V |
| defer | D, E1, F, ER, R |
| deficit | D, EH1, F, I3, S, I1, T |
| degree | D, Y, G, R, E1, Y |

| Word | Phonetic Program |
|---|---|
| delay | D, I1, L, EH3, A1, Y |
| delete | D, E1, L, E1, Y, T |
| deliver | D, Y, L, I1, V, ER |
| delta | D, EH2, EH3, L, T, UH1 |
| demand | D, Y, M, AE1, EH3, N, D |
| demonstrate | D, EH1, M, UH3, N, S, T, R, A1, Y, T |
| deny | D, Y, N, AH1, EH3, Y |
| destroy | D, Y, S, T, R, O1, UH3, I3, AY |
| detail | D, E, T, EH3, A1, I3, UH3, L |
| determine | D, Y, T, ER, M, I1, N |
| device | D, Y, V, UH3, AH2, Y, S |
| dew | (use "do" program) |
| diagnostic | D, AH1, AY, I3, G, N, AH1, UH3, S, T, I3, K |
| dial | D, AH1, EH3, I3, UH3, L |
| dictionary | D, I1, I3, K, SH, UH3, N, EH3, EH3, ER, Y |
| did | D, I1, I3, D |
| die | D, AH1, EH3, Y |
| diet | D, AH1, EH3, AY, I2, T |
| differ | D, I1, I3, F, ER |
| difference | D, I1, F, R, EH3, N, DT, S |
| different | D, I1, F, R, EH3, N, T |
| digit | D, I1, D, J, I1, T |
| digital | D, I1, D, J, I3, T, UH3, L |
| dime | D, AH1, EH3, Y, M |
| diode | D, AH1, EH3, AY, O1, U1, D |
| direct | D, ER, EH1, EH3, K, T |
| directory | D, ER, EH1, EH3, K, T, ER, Y |
| dirt | D, ER, R, T |
| disagree | D, I1, S, UH1, G, R, E1, Y |
| disappear | D, I1, S, UH1, P, AY, I3, R |
| disconnect | D, I1, S, K, UH1, N, EH1, EH3, K, T |
| discuss | D, I1, I3, S, K, UH1, UH2, S |
| disk | D, I1, I3, S, K |
| display | D, I1, I3, S, P, L, A1, I3, Y |
| distance | D, I1, S, T, EH3, N, T, S |
| divide | D, I1, V, AH1, EH3, Y, D |
| dividend | D, I1, V, I1, D, EH1, EH3, N, D |
| division | D, I1, V, I1, ZH, UH3, N |
| do | D, IU, U1, U1 |
| dock | D, AH1, UH3, K |
| doctor | D, AH1, UH3, K, T, ER |
| document | D, AH1, K, Y1, UH3, M, EH3, N, T |
| does | D, UH2, UH1, Z |
| dollar | D, AH1, UH3, L, ER |
| done | D, UH1, UH3, N |

| Word | Phonetic Program |
|---|---|
| door | D, O1, O2, R |
| double | D, UH3, UH1, B, UH3, L |
| doubt | D, UH3, AH2, U1, T |
| down | D, AH1, UH3, U1, N |
| draft | D, R, AE1, EH3, F, T |
| draw | D, R, AW |
| drill | D, R, I1, I3, L |
| drink | D, R, I1, I3, NG, K |
| drive | D, R, AH1, EH3, Y, V |
| drop | D, R, AH1, UH3, P |
| drum | D, R, UH1, UH2, M |
| dry | D, R, AH1, EH3, I3, Y |
| due | (use "do" program) |
| dump | D, UH1, UH2, M, P |
| duration | D, ER, R, A1, Y, SH, UH3, N |
| during | D, ER, R, I1, NG |
| duty | D, IU, U1, U1, T, Y |
| dwell | D, W, EH1, EH3, L |
| E | E1, Y |
| each | E1, AY, T, CH |
| ear | E1, I2, R |
| early | ER, R, L, Y |
| earn | ER, R, N |
| east | E1, AY, S, T |
| easy | E1, AY, Z, Y |
| echo | EH1, EH3, K, O1, U1 |
| edge | EH1, EH3, D, J |
| edit | EH1, EH3, D, I2, T |
| educate | EH1, D, J, U1, K, A1, Y, T |
| effect | UH1, F, EH1, EH3, K, T |
| efficient | E1, F, I1, SH, EH3, N, T |
| effort | EH2, EH3, F, ER, T |
| eight | A2, A2, Y, T |
| eighth | A2, A2, Y, DT, DT, TH |
| eighty | A2, A2, Y, T, Y |
| either | E1, Y, THV, ER |
| electric | EH3, L, EH1, K, T, R, I2, K |
| electrician | EH3, L, EH1, K, PAO, T, R, I1, SH, UH3, N |
| electronic | EH3, L, EH1, K, T, R, AH1, N, I2, K |
| elevator | EH1, L, UH3, V, A2, AY, D, ER |
| eleven | EH1, L, EH1, EH3, V, I2, N |
| eligible | EH1, L, UH3, D, J, EH3, B, UH3, L |
| eliminate | EH1, L, I1, M, I1, N, A1, Y, T |
| else | EH1, EH3, L, S |
| emit | Y, M, I1, I3, T |
| employ | EH1, EH3, M, P, L, O1, UH3, I3, AY |
| empty | EH1, EH3, M, P, T, Y |
| enable | EH1, N, A1, Y, B, UH3, L |
| enclose | EH1, EH3, N, K, L, O1, U1, Z |
| end | EH1, EH3, N, D |
| engine | EH1, EH3, N, D, J, I1, N |
| engineer | EH1, N, D, J, I2, N, AY, I1, R |
| endorse | EH1, EH3, N, D, O2, O2, R, S |
| english | I1, NG, G, L, I2, SH |
| enter | EH1, EH3, N, T, ER |
| entry | EH1, EH3, N, T, R, Y |
| epsilon | EH1, P, S, UH3, L, AH1, UH3, N |
| equal | Y, K, W, UH3, L |
| equipment | E1, K, W, IL, P, M, EH3, N, T |
| erase | E1, R, A1, Y, S |
| error | EH3, EH3, EH3, R, ER |
| escape | EH1, EH3, S, K, A1, AY, Y, P |
| escrow | EH1, EH3, S, K, R, O1, U1 |
| establish | UH1, S, T, AE1, EH3, B, L, I2, SH |
| estate | EH1, EH3, S, T, A1, AY, Y, T |
| estimate | EH1, S, T, EH3, M, I3, T |
| exact | EH1, EH3, G, PAO, Z, AE1, EH3, K, T |
| examine | EH1, EH3, G, PAO, Z, AE1, EH3, M, I1, N |
| exceed | EH1, EH3, K, PAO, S, E1, Y, D |
| except | EH1, EH3, K, PAO, S, EH1, EH3, P, T |
| exchange | EH1, EH3, K, PAO, S, T, CH, A1, AY, Y, N, D, J |
| execute | EH1, EH3, K, PAO, S, UH3, K, Y1, IU, U1, T |
| exempt | EH1, EH3, G, PAO, Z, EH1, EH3, M, P, T |
| exit | EH1, EH3, G, PAO, Z, I1, I3, T |
| expect | EH1, EH3, K, PAO, S, P, EH1, EH3, K, T |
| expedite | EH1, EH3, K, PAO, S, P, EH1, EH3, D, UH3, AH2, Y, T |
| expend | EH1, EH3, K, PAO, S, P, EH1, EH3, N, D |
| experiment | EH1, K, PAO, S, P, EH1, R, UH3, M, EH3, N, T |
| exponent | EH1, K, PAO, S, P, O2, O2, N, EH3, N, T |
| express | EH1, EH3, K, PAO, S, P, R, EH1, S |
| extension | EH1, EH3, K, PAO, S, T, EH1, EH3, N, SH, UH3, N |
| F | EH1, EH2, F |

| Word | Phonetic Program |
|---|---|
| face | F, A1, AY, Y, S |
| facility | F, UH2, S, I1, L, I3, T, Y |
| fact | F, AE1, EH3, K, T |
| fahrenheit | F, EH1, R, I2, N, H, UH3, AH2, Y, T |
| fail | F, A1, AY, I3, UH3, L |
| fall | F, AW, L |
| false | F, AW, L, S |
| familiar | F, UH1, M, I1, L, Y1, ER |
| far | F, AH1, UH3, R |
| farad | F, EH3, EH3, ER, AE1, EH3, D |
| fast | F, AE1, EH3, S, T |
| fault | F, AW, L, T |
| feat | (use "feet" program) |
| feature | F, E1, AY, T, CH, ER |
| february | F, EH1, B, Y1, IU, W, EH1, R, Y |
| federal | F, EH1, EH3, D, R, UH3, L |
| fee | F, E1, Y |
| feed | F, E1, Y, D |
| feet | F, E1, Y, T |
| female | F, AY, Y, M, A1, AY, UH3, L |
| field | F, E1, AY, UH3, L, D |
| fifteen | F, I1, I3, F, T, E1, Y, N |
| fifth | F, I1, I3, F, TH |
| fifty | F, I1, I3, F, T, Y |
| file | F, AH1, EH3, I3, UH3, L |
| fill | F, I1, I3, L |
| final | F, AH1, Y, N, UH3, L |
| finance | F, AH1, EH3, Y, N, AE1, EH3, N, S |
| find | F, AH1, EH3, Y, N, D |
| finger | F, I1, I3, NG, G, ER |
| finish | F, I1, N, I1, SH |
| fire | F, AH1, EH3, AY, R |
| first | F, ER, R, S, T |
| fit | F, I1, I3, T |
| five | F, AH1, EH3, Y, V |
| fix | F, I1, I3, K, PAO, S |
| fixture | F, I1, I3, K, PAO, S, T, CH, ER |
| flash | F, L, AE1, EH3, SH |
| flat | F, L, AE1, EH3, T |
| flight | F, L, UH3, AH2, Y, T |
| flip | F, L, I1, I3, P |
| floor | F, L, O1, O2, R |
| flop | F, L, AH1, UH3, P |
| flow | F, L, O1, U1 |
| fly | F, L, AH1, EH3, Y |
| fold | F, O2, O2, L, L, D |
| follow | F, AH1, AW2, L, O1, U1 |
| food | F, U1, U1, D |
| foot | F, OO1, OO1, T |
| for | (use "four" program) |
| fore | (use "four" program) |
| force | F, O2, O2, R, S |
| foreman | F, O2, O2, R, M, EH2, N |
| forget | F, O2, O2, R, G, EH1, EH3, T |
| forgive | F, O2, O2, R, G, I1, I3, V |
| form | F, O2, O2, R, M |
| format | F, O2, O2, R, M, AE1, EH3, T |
| forty | F, O2, O2, R, T, Y |
| forward | F, O2, O2, R. W, ER, D |
| found | F, AH1, UH3, W, N, D |
| four | F, O1, O2, R |
| fourth | F, O1, O2, R, TH |
| fox trot | F, AH1, UH3, K, PAO, S, T, R, AH1, UH3, T |
| frame | F, R, A1, AY, Y, M |
| fraud | F, R, AW, D |
| free | F, R, E1, Y |
| french | F, R, EH1, EH3, N, T, CH |
| frequency | F, R, E1, K, W, EH3, N, DT, S, Y |
| frequent | F, R, E1, K, W, EH3, N, T |
| friday | F, R, AH1, EH3, Y, D, A1, I3, Y |
| fright | F, R, UH3, AH2, Y, T |
| from | F, R, UH1, UH3, M |
| front | F, R, UH3, UH1, N, T |
| fruit | F, R, IU, U1, T |
| fuel | F, Y1, IU, U1, UH3, L |
| full | F, OO1, L |
| function | F, UH1, UH2, N, K, SH, UH3, N |
| fund | F, UH1, UH2, N, D |
| furnace | F, ER, R, N, EH3, S |
| further | F, ER, R, THV, ER |
| future | F, Y1, IU, U1, T, CH, ER |
| G | D, J, E1, Y |
| gage | (use "gauge" program) |
| gain | G, A1, AY, Y, N |
| gait | (use "gate" program) |
| gallon | G, AE1, AH2, L, UH3, N |
| game | G, A1, AY, Y, M |
| gamma | G, AE1, EH3, M, UH2, UH3 |
| gap | G, AE1, EH3, P |
| garage | G, UH1, R, AH1, UH3, ZH |
| gas | G, AE1, EH3, S |
| gate | G, A1, AY, Y, T |
| gauge | G, A1, AY, Y, D, J |
| general | D, J, EH1, EH3, N, ER, UH3, L |
| generate | D, J, EH1, N, ER, A1, Y, T |
| gentlemen | D, J, EH1, EH3, N, T, L, M, I2, N |

| Word | Phonetic Program |
|---|---|
| german | D, J, ER, R, M, EH2, N |
| get | G, EH1, EH3, T |
| girl | G, ER, R, L |
| give | G, I1, I3, V |
| glass | G, L, AE1, EH3, S |
| glitch | G, L, I1, I3, T, CH |
| globe | G, L, O1, U1, B |
| go | G, OO1, O1, U1 |
| golf | G, AW2, AW2, UH3, L, F |
| good | G, OO1, OO1, D |
| govern | G, UH1, UH3, V, ER, N |
| grade | G, R, A1, AY, Y, D |
| gram | G, R, AE1, EH3, M |
| grand | G, R, AE1, EH3, N, D |
| graph | G, R, AE1, EH3, F |
| grate | (use "great" program) |
| gray | (use "grey" program) |
| great | G, R, A1, Y, T |
| green | G, R, E1, Y, N |
| greet | G, R, E1, Y, T |
| grey | G, R, A1, AY, Y |
| grind | G, R, AH1, EH3, Y, N, D |
| grocery | G, R, O1, U1, S, ER, Y |
| ground | G, R, AH1, UH3, W, N, D |
| group | G, R, U1, U1, P |
| grow | G, R, O1, U1 |
| guard | G, AH1, R, D |
| guarantee | G, EH1, R, I3, N, T, E1, Y |
| guess | G, EH1, EH3, S |
| H | A1, AY, Y, T, CH |
| had | H, AE1, EH3, D |
| half | H, AE1, EH3, F |
| halt | H, AW, L, T |
| hammer | H, AE1, EH3, M, ER |
| hand | H, AE1, EH3, N, D |
| handle | H, AE1, EH3, N, D, UH3, L |
| hang | H, AE1, I3, NG |
| happy | H, AE1, EH3, P, Y |
| hard | H, AH1, R, D |
| has | H, AE1, EH3, Z |
| have | H, AE1, EH3, V |
| he | H, E1, Y |
| head | H, EH1, EH3, D |
| hear | H, AY, I3, R |
| heart | H, AH1, UH3, R, T |
| heat | H, E1, AY, T |
| heavy | H, EH1, V, Y |
| height | H, UH3, AH2, Y, T |
| held | H, EH1, UH3, L, D |
| hello | H, EH1, UH3, L, UH3, O1, U1 |
| help | H, EH1, EH3, L, P |
| henry | H, EH1, EH3, N, R, Y |
| her | H, ER |
| here | (use "hear" program) |
| hertz | H, R, R, T, S |
| hex | H, EH1, EH3, K, PAO, S |
| high | H, AH1, EH3, Y |
| his | H, I1, I3, Z |
| hold | H, O2, O2, L, L, D |
| hole | H, O1, U1, L |
| home | H, O1, U1, M |
| hook | H, OO1, OO1, K |
| host | H, O1, U1, S, T |
| hot | H, AH1, UH3, T |
| hotel | H, O1, U1, T, EH2, EH2, L |
| hour | AH1, UH3, W, ER |
| house | H, UH3, AH2, U1, S |
| how | H, AH1, O2, U1 |
| human | H, Y1, IU, U1, U1, M, EH2, N |
| hundred | H, UH1, UH2, N, D, R, I3, D |
| hungry | H, UH1, UH2, NG, G, R, Y |
| I | AH1, EH3, I3, Y |
| idle | AH1, Y, D, UH3, L |
| idol | (use "idle" program) |
| if | I1, I3, F |
| immediate | I1, I3, M, E1, D, Y, EH3, T |
| important | I1, I3, M, P, O2, O2, R, T, EH3, N, T |
| improper | I1, I3, M, P, R, AH1, UH3, P, ER |
| improve | I1, I3, M, P, R, IU, U1, U1, V |
| in | I1, I3, N |
| inch | I1, I3, N, T, CH |
| include | I1, I3, N, K, L, IU, U1, U1, D |
| income | I1, I3, N, K, UH1, UH3, M |
| independent | I1, N, D, E1, P, EH2, EH3, N, D, EH3, N, T |
| index | I1, I3, N, D, EH1, EH3, K, PAO, S |
| india | I2, I3, N, D, Y, UH2 |
| indicate | I1, N, D, I3, K, A1, Y, T |
| industrial | I1, I3, N, D, UH1, UH2, S, T, R, AY, UH3, L |
| inform | I1, I3, N, F, O2, O2, R, M |
| initial | I1, I3, N, I1, SH, UH3, L |
| inn | (use "in" program) |
| input | I1, I3, N, P, OO1, OO1, T |
| inquire | I1, I3, N, K, W, AH1, EH3, AY, R |
| insert | I1, N, S, R, R, T |
| inspect | I1, I3, N, S, P, EH1, EH3, K, T |

| Word | Phonetic Program |
|---|---|
| install | I1, I3, N, S, T, AW, L |
| instead | I1, I3, N, S, T, EH1, EH3, D |
| instruct | I1, I3, N, S, T, R, UH1, UH2, K, T |
| instrument | I1, I3, N, S, T, R, UH1, M, EH1, EH3, N, T |
| insufficient | I1, N, S, UH2, F, I1, SH, EH3, N, T |
| insurance | I1, I3, N, SH, ER, R, EH3, N, T, S |
| interest | I1, N, T, R, EH1, S, T |
| interface | I1, I3, N, T, ER, F, A1, AY, Y, S |
| interpret | I1, I3, N, T, ER, P, R, EH3, T |
| interupt | I1, N, T, ER, UH3, UH1, P, T |
| intrude | I1, I3, N, T, R, IU, U1, U1, D |
| invalid | I1, I3, N, V, AE1, AW2, L, I1, D |
| invent | I1, I3, N, V, EH1, EH3, N, T |
| inventory | I1, N, V, EH1, N, T, O1, R, Y |
| invest | I1, I3, N, V, EH1, EH3, S, T |
| invoice | I1, I3, N, V, O1, UH3, I3, AY, S |
| irregular | I1, R, EH1, G, Y1, UH3, L, ER |
| is | I1, I3, Z |
| it | I1, I3, T |
| item | AH2, UH3, Y, D, UH3, M |
| J | D, J, EH3, A1, AY, Y |
| jack | D, J, AE1, EH3, K |
| january | D, J, AE1, EH3, N, Y1, UI, EH3, EH3, ER, Y |
| job | D, J, AH1, UH3, B |
| join | D, J, O1, UH3, I3, AY, N |
| jolt | D, J, O2, O2, L, T |
| joy | D, J, O1, UH3, I3, AY |
| judge | D, J, UH1, UH2, D, J |
| juliet | D, J, IU, U1, L, Y, EH2, EH3, T |
| july | D, J, UH1, L, AH1, EH3, Y |
| jump | D, J, UH1, UH2, M, P |
| june | D, J, IU, U1, U1, N |
| K | K, EH3, A1, AY, Y |
| keep | K, E1, Y, P |
| key | K, E1, Y |
| keyboard | K, AY, Y, B, O1, O2, R, D |
| kill | K, I1, I3, L |
| kilo | K, E1, AY, L, UH3, O2, U1 |
| knew | (use "new" program) |
| knot | (use "not" program) |
| know | (use "no" program) |
| knowledge | N, AH1, UH3, L, I3, D, J |
| L | EH1, EH3, UH3, L |
| lab | L, AE1, EH3, B |
| labor | L, A1, Y, B, ER |
| language | L, AE1, EH3, NG, G, W, I1, D, J |
| lapse | L, AE1, EH3, P, S |
| large | L, AH1, R, D, J |
| last | L, AE1, EH3, S, T |
| late | L, A1, AY, Y, T |
| law | L, AW |
| lead | L, E1, Y, D |
| led | L, EH1, EH3, D |
| left | L, EH1, EH3, F, T |
| leg | L, EH1, EH3, G |
| legal | L, E1, G, UH3, L |
| lend | L, EH1, EH3, N, D |
| length | L, EH1, EH3, NG, TH |
| less | L, EH1, EH3, S |
| let | L, EH1, EH3, T |
| letter | L, EH1, EH3, T, ER |
| level | L, EH1, EH3, V, UH3, L |
| life | L, UH3, AH2, Y, F |
| light | L, UH3, AH2, Y, T |
| like | L, UH3, AH2, Y, K |
| lima | L, AY, Y, M, UH1 |
| limit | L, I1, M I1, T |
| line | L, AH1, EH3, Y, N |
| linear | L, I2, I3, N, AY, Y, ER |
| link | L, I1, I3, NG, K |
| lip | L, I1, I3, P |
| liquid | L, I1, K, W, I1, D |
| list | L, I1, I3, S, T |
| listen | L, I1, I3, S, I2, N |
| little | L, I1, I3, T, UH3, L |
| load | L, UH3, 01, U1, D |
| loan | L, UH3, 01, U1, N |
| local | L, 02, 02, K, UH3, L |
| lock | L, AH1, UH3, K |
| log | L, AW, G |
| long | L, AW, NG |
| look | L, 001, 001, K |
| loss | L, AW, S |
| lost | L, AW, S, T |
| lot | L, AH1, UH3, T |
| low | L, 01, U1 |
| M | EH1, EH2, M |
| machine | M, UH2, SH, E1, Y, N |
| mail | (use "male" program) |
| maintenance | M, A1, Y, N, T, EH2, N, EH3, N, DT, S |
| make | M, A1, AY, Y, K |
| male | M, A2, A2, AY UH3, L |
| man | M, AE1, EH3, N |

| Word | Phonetic Program |
|---|---|
| manage | M, AE1, EH3, N, I1, D, J |
| manual | M, AE1, EH3, N, Y1, U1, UH3, L |
| manufacture | M, AE1, EH3, N, Y1, U1, F, AE1, EH3, K, T, CH, ER |
| many | M, EH2, EH2, N, Y |
| map | M, AE1, EH3, P |
| march | M, AH1, R, T, CH |
| margin | M, AH1, UH3, R, D, J, I2, N |
| mark | M, AH1, R, K |
| market | M, AH1, R, K, EH3, T |
| match | M, AE1, EH3, T, CH |
| mature | M, UH1, T, CH, IU, ER |
| maximum | M, AE1, EH3, K, PAO, S, EH3, M, UH2, M |
| may | M, A1, I3, Y |
| me | M, E1, Y |
| measure | M, EH3, EH1, ZH, ER |
| meat | M, E1, AY, T |
| mechanical | M, UH1, K, AE1, EH3, N, I3, K, UH3, L |
| media | M, E1, AY, D, Y, UH1 |
| medicine | M, EH2, EH3, D, I3, S, I1, N |
| medium | M, E1, D, AY, UH1, M |
| meet | (use "meat" program) |
| mega | M, EH1, EH3, G, UH2, UH3 |
| member | M, EH1, EH3, M, B, ER |
| memory | M, EH1, EH3, M, ER, Y |
| men | M, EH1, EH3, N |
| merchandise | M, ER, T, CH, EH3, N, D, AH1, EH3, Y, Z |
| merge | M, ER, R, D, J |
| message | M, EH1, EH3, S, I2, D, J |
| metal | M, EH1, EH3, T, UH3, L |
| meter | M, E1, Y, T, ER |
| micro | M, UH3, AH2, AY, K, R, 01, U1 |
| middle | M, I1, I3, D, UH3, L |
| mike | M, UH3, AH2, Y, K |
| mile | M, AH1, EH3, I3, UH3, L |
| mill | M, I1, I3, L |
| milli | M, I1, I3, L, UH3 |
| million | M, I1, I3, L, Y, UH3, N |
| mini | M, I2, I2, N, Y |
| minus | M, AH1, Y, N, EH3, S |
| minute | M, I1, N, EH3, T |
| miscellaneous | M, I1, S, UH3, L, A1, AY, N, Y, UH3, S |
| miss | M, I1, I3, S |
| mistake | M, I1, I3, S, T, A1, AY, Y, K |
| mode | M, 01, U1, D |
| model | M, AH1, UH3, D, UH3, L |
| module | M, AH1, UH3, D, J, IU, U1, UH3, L |
| monday | M, UH3, UH1, N, D, A1, I3, Y |
| money | M, UH3, UH1, N, AY, Y |
| month | M, UH3, UH1, N, TH |
| more | M, 02, 02, R |
| morning | M, 02, 02, R, N, I1, I3, NG |
| most | M, 01, U1, S, T |
| motor | M, 01, U1, T, ER |
| mount | M, AH1, UH3, W, N, T |
| move | M, U1, U1, V |
| Mr. | M, I1, S, T, ER |
| Mrs. | M, I1, S, I2, Z |
| Ms. | M, I1, I3, Z |
| much | M, UH1, UH2, T, CH |
| multi | M, UH2, UH3, L, T, Y |
| multiple | M, UH1, L, T, EH3, P, UH3, L |
| multiply | M, UH1, L, T, I3, P, L, AH1, Y |
| N | EH1, EH2, N |
| name | N, A1, AY, Y, M |
| nano | N, AE1, EH3, N, 01, U1 |
| national | N, AE1, EH3, SH, UH3, N, UH3, L |
| native | N, A1, Y, T, I1, V |
| near | N, AY, I1, R |
| neat | N, E1, AY, T |
| neck | N, EH1, EH3, K |
| need | N, E1, Y, D |
| negative | N, EH1, G, EH3, T, I1, V |
| net | N, EH1, EH3, T |
| neutral | N, IU, U1, T, R, UH2, L |
| new | N, IU, U1, U1 |
| next | N, EH1, EH3, K, PAO, S, T |
| nice | N, UH3, AH2, Y, S |
| nickel | N, I1, I3, K, UH3, L |
| night | N, UH3, AH2, Y, T |
| nine | N, AH1, EH3, Y, N |
| ninety | N, AH1, EH3, Y, N, T, Y |
| nineth | N, AH1, Y, N, DT, TH |
| no | N, 001, 01, U1 |
| noise | N, 01, UH3, I3, AY, Z |
| none | N, UH1, UH3, N |
| noon | N, IU, U1, U1, N |
| normal | N, 02, 02, R, M, UH3, L |
| north | N, 02, 02, R, TH |
| not | N, AH1, UH3, T |
| note | N, 01, U1, T |
| nothing | N, UH1, TH, I1, I3, NG |
| notice | N, 01, U1, T, I1, S |

| Word | Phonetic Program |
|---|---|
| notify | N, 01, U1, T, I1, F, AH1, EH3, Y |
| november | N, 01, U1, V, EH1, EH3, M, B, ER |
| now | N, AH1, UH3, U1 |
| number | N, UH1, UH2, M, B, ER |
| nurse | N, ER, R, S |
| nut | N, UH1, UH2, T |
| O | 02, 01, U1 |
| oar | (use "or" program) |
| object | UH1, B, D, J, EH1, EH3, K, T |
| object-2 | AH1, UH3, B, D, J, EH2, EH2, K, T |
| obligation | AH1, B, L, I3, G, A1, Y, SH, UH3, N |
| obsolete | AH1, UH3, B, S, UH3, L, AY, Y, T |
| october | AH1, UH3, K, T, O1, U1, B, ER |
| odd | AH1, UH3, D |
| of | UH1, UH3, V |
| off | AW, F |
| office | AW, F, I1, S |
| official | UH1, F, I1, SH, UH3, L |
| often | AW2, AW2, F, I3, N |
| ohm | O2, O2, U1, M |
| oil | O1, EH3, I3, UH3, L |
| old | O2, O2, L, L, D |
| omega | O1, U1, M, A1, Y, G, UH2 |
| omit | O1, U1, M, I1, I3, T |
| on | AH1, UH3, N |
| once | W, UH1, N, T, S |
| one | W, UH1, UH2, N |
| only | O1, O2, N, L, Y |
| open | O1, P, I2, N |
| operable | AH1, UH3, P, ER, UH3, B, UH3, L |
| operate | AH1, UH3, P, ER, A1, Y, T |
| operator | AH1, UH3, P, ER, A1, Y, T, ER |
| option | AH1, UH3, P, SH, UH3, N |
| or | O2, O2, R |
| orange | O2, O2, R, I1, N, D, J |
| order | O2, O2, R, D, ER |
| ore | (use "or" program) |
| original | O2, R, I2, I3, D, J, I3, N, UH3, L |
| oscar | AH1, UH3, S, K, ER |
| other | UH1, UH3, THV, ER |
| ounce | AH1, UH3, W, N, S |
| out | UH3, AH2, U1, T |
| oven | UH1, V, I2, N |
| over | O1, O2, V, ER |
| oxygen | AH1, UH3, K, PAO, S, I3, D, J, I2, N |
| own | O1, U1, N |
| P | P, E1, Y |
| pack | P, AE1, EH3, K |
| package | P, AE1, EH3, K, I1, D, J |
| paid | P, A1, AY, Y, D |
| pain | P, A1, AY, Y, N |
| pane | (use "pain" program) |
| panel | P, AE1, EH3, N, UH3, L |
| papa | P, AH1, UH3, P, UH3, UH3 |
| paper | P, A1, Y, P, ER |
| parcel | P, AH1, R, S, UH3, L |
| paren | P, EH3, EH3, ER, I2, N |
| part | P, AH1, R, T |
| partial | P, AH1, R, SH, UH2, L |
| pass | P, AE1, EH3, S |
| passed | (use "past" program) |
| past | P, AE1, EH3, S, T |
| pat | P, AE1, EH3, T |
| pattern | P, AE1, EH3, T, ER, N |
| pause | P, AW, Z |
| pay | P, A2, A2, AY, Y |
| pea | (use "P" program) |
| peace | (use "piece" program) |
| peak | P, E1, AY, K |
| peek | (use "peak" program) |
| percent | P, ER, S, EH1, EH3, N, T |
| period | P, I1, R, Y, UH2, D |
| permanent | P, ER, M, EH2, N, EH1, N, T |
| person | P, ER, S, UH1, N |
| personal | P, ER, S, UH3, N, UH2, L |
| personality | P, ER, S, UH3, N, AE1, UH3, L, I3, T, Y |
| phase | F, A1, AY, Y, Z |
| phone | F, O1, U1, N |
| pick | P, I1, I3, K |
| pico | P, E1, Y, K, O2, U1 |
| piece | P, E1, Y, S |
| pint | P, AH1, Y, N, T |
| pipe | P, UH3, AH2, Y, P |
| place | P, L, A1, AY, Y, S |
| plain | (use "plane" program) |
| plan | P, L, AE1, EH3, N |
| plane | P, L, A1, AY, Y, N |
| plant | P, L, AE1, EH3, N, T |
| play | P, L, A1, I3, Y |
| please | P, L, E1, Y, Z |
| plot | P, L, AH1, UH3, T |
| plus | P, L, UH1, UH2, S |
| pocket | P, AH1, UH3, K, EH3, T |
| point | P, O1, UH3, I3, AY, N, T |

| Word | Phonetic Program |
|---|---|
| poke | P, O1, U1, K |
| police | P, UH1, L, AY, Y, S |
| plain | (use "plane" program) |
| plan | P, L, AE1, EH3, N |
| plane | P, L, A1, AY, Y, N |
| plant | P, L, AE1, EH3, N, T |
| play | P, L, A1, I3, Y |
| please | P, L, E1, Y, Z |
| plot | P, L, AH1, UH3, T |
| plus | P, L, UH1, UH2, S |
| pocket | P, AH1, UH3, K, EH3, T |
| point | P, O1, UH3, I3, AY, N, T |
| poke | P, O1, U1, K |
| police | P, UH1, L, AY, Y, S |
| policy | P, AH1, UH3, L, I3, S, Y |
| poor | (use "pour" program) |
| pop | P, AH1, UH3, P |
| port | P, 02, 02, R, T |
| position | P, UH1, Z, I1, SH, UH3, N |
| positive | P, AH1, UH3, Z, I1, T, I1, V |
| possible | P, AH1, UH3, S, UH3, B, UH2, L |
| post | P, 01, U1, S, T |
| potential | P, 01, T, EH1, EH3, N, T, CH, UH3, L |
| pound | P, AH1, UH3, W, N, D |
| pour | P, 01, 02, R |
| power | P, AH1, UH3, W, ER |
| practice | P, R, AE1, EH3, K, T, I1, S |
| premium | P, R, AY, Y, M, Y, UH1, M |
| prepare | P, R, E1, P, EH1, EH3, R |
| press | P, R, EH1, EH3, S |
| pressure | P, R, EH1, SH, ER |
| prevent | P, R, Y, V, EH1, EH3, N, T |
| previous | P, R, Y, V, Y, UH1, S |
| price | P, R, UH3, AH2, Y, S |
| principal | (use "principle" program) |
| principle | P, R, I1, N, DT, S, UH3, P, UH3, L |
| print | P, R, I1, I3, N, T |
| prior | P, R, AH1, Y, ER |
| priority | P, R, AH1, Y, 01, R, I3, DT, Y |
| private | P, R, AH1, EH3, Y, V, I3, T |
| probe | P, R, 01, U1, B |
| problem | P, R, AH1, UH3, B, L, UH3, M |
| procedure | P, R, UH1, S, E1, D, J, ER |
| proceed | P, R, 01, S, E1, Y, D |
| process | P, R, AH1, UH3, S, EH1, EH3, S |
| produce | P, R, UH1, D, IU, U1, U1, S |
| product | P, R, AH1, UH3, D, UH1, UH2, K, T |
| progress | P, R, AH1, UH3, G, R, EH1, S |
| profession | P, R, UH1, F, EH1, EH3, SH, UH3, N |
| profit | P, R, AH1, UH3, F, I1, T |
| program | P, R, 01, G, R, AE1, EH3, M |
| project | P, R, AH1, UH3, D, J, EH2, EH2, K, T |
| PROM | P, R, AH1, UH3, M |
| promote | P, R, UH1, M, 01, U1, T |
| propose | P, R, UH1, P, 01, U1, Z |
| protect | P, R, UH1, T, EH1, EH3, K, T |
| public | P, UH1, UH3, B, L, I3, K |
| pull | P, 001, 001, L |
| pulse | P, UH1, UH2, L, S |
| punch | P, UH1, UH2, N, T, CH |
| purpose | P, R, R, P, EH2, S |
| purchase | P, R, R, DT, CH, I2, S |
| pure | P, Y1, IU, ER |
| push | P, 001, IU, SH |
| put | P, 001, 001, T |
| Q | K, Y1, IU, U1, U1 |
| qualify | K, W, AW1, L, I1, F, AH1, EH3, Y |
| quantity | K, W, AH1, N, T, I3, T, Y |
| quart | K, W, 01, R, T |
| quarter | K, W, 01, R, T, ER |
| quebec | K, W, I1, B, EH1, EH3, K |
| question | K, W, EH1, EH3, S, T, CH, UH3, N |
| quick | K, W, I1, I3, K |
| quiet | K, W, AH1, EH3, AY, I2, T |
| quit | K, W, I1, I3, T |
| quiz | K, W, I1, I3, Z |
| quota | K, W, 01, 02, T, UH1 |
| quote | K, W, 01, U1, T |
| R | AH1, UH2, ER |
| rail | R, A1, AY, I3, UH3, L |
| rain | R, A1, AY, Y, N |
| raise | R, A1, AY, Y, Z |
| range | R, A1, AY, Y, N, D, J |
| radio | R, A1, Y, D, Y, 01, U1 |
| rate | R, A1, AY, Y, T |
| ratio | R, A1, Y, SH, Y, 01, U1 |
| reach | R, E1, Y, T, CH |
| read | R, E1, Y, D |
| ready | R, EH1, EH3, D, Y |
| real | R, E1, AY, L |
| reason | R, E1, Y, Z, UH1, N |
| rebate | R, E1, B, A1, Y, T |

| Word | Phonetic Program |
|---|---|
| recall | R, E1, K, AW2, AW1, L |
| receipt | R, E1, S, AY, Y, T |
| receive | R, E1, S, E1, Y, V |
| record | R, E1, K, 02, 02, R, D |
| record-2 | R, EH1, EH3, K, ER, D |
| red | R, EH1, EH3, D |
| reel | (use "real" program) |
| refer | R, E1, F, UH1, UH2, N, D |
| refuse | R, E1, F, Y1, IU, U1 U1, Z |
| register | R, EH1, D, J, I1, S, T, ER |
| regular | R, EH1, G, Y1, IU, L, ER |
| rein | (use "rain" program) |
| reject | R, E1, D, J, EH1, EH3, K, T |
| relay | R, E1, L, A1, I3, Y |
| release | R, E1, L, E1, AY, S |
| remain | R, E1, M, A1, AY, Y, N |
| remove | R, E1, M, U1, U1, V |
| repair | R, E1, P, EH2, EH2, R |
| repeat | R, E1, P, E1, AY, T |
| replace | R, E1, P, L, A1, AY, Y, S |
| report | R, E1, P, 02, 02, R, T |
| represent | R, EH1, P, R, I2, Z, EH1, EH3, N, T |
| request | R, E1, K, W, EH1, EH3, S, T |
| require | R, E1, K, W, AH1, EH3, AY, R |
| requisition | R, EH1, K, W, I2, Z, I1, SH, UH3, N |
| rescue | R, EH1, EH3, S, K, Y1, IU, U1 |
| resemble | R, E1, Z, EH1, EH3, M, B, UH3, L |
| reset | R, E1, S, EH1, EH3, T |
| resistor | R, E1, Z, I1, S, T, ER |
| respect | R, E1, S, P, EH1, EH3, K, T |
| respond | R, E1, S, P, AH1, UH3, N, D |
| responsible | R, I2, S, P, AH1, UH3, N, DT, S, UH3, B, UH3, L |
| rest | R, EH1, EH3, S, T |
| restrict | R, E1, S, T, R, I1, I3, K, T |
| result | R, E1, Z, UH1, UH2, L, T |
| resume | R, E1, Z, IU, U1, U1, M |
| retail | R, AY, E1, T, EH3, A1, I3, UH3, L |
| retain | R, E1, T, A1, AY, Y, N |
| return | R, E1, T, ER, R, N |
| revision | R, E1, V, I1, ZH, UH3, N |
| revolve | R, E1, V, AH1, UH3, L, V |
| ribbon | R, I2, I3, B, UH3, N |
| right | R, UH3, AH2, Y, T |
| romeo | R, 01, U1, M, Y, 01, U1 |
| room | R, U1, U1, M |
| root | R, U1, U1, T |
| round | R, AH1, UH3, W, N, D |
| route | R, UH2, AH2, U1, T |
| row | R, 01, U1 |
| run | R, UH1, UH3, N |
| rush | R, UH1, UH2, SH |
| S | EH1, EH2, S |
| safe | S, A1, AY, Y, F |
| sail | (use "sale" program) |
| salary | S, AE1, AH2, L, UH3, R, Y |
| sale | S, A1, A2, AY, UH3, L |
| same | S, A1, AY, Y, M |
| saturday | S, AE1, EH3, T, ER, D, A1, Y |
| save | S, A1, AY, Y, V |
| say | S, A1, I3, Y |
| scan | S, K, AE1, EH3, N |
| scent | (use "cent" program) |
| schedule | S, K, EH1, EH3, D, J, IU, U1, L |
| school | S, K, U1, U1, L |
| science | S, AH1, I3, Y, EH3, N, DT, S |
| score | S, K, O2, O2, R |
| scrap | S, K, R, AE1, EH3, P |
| screw | S, K, R, IU, U1, U1 |
| sea | (use "C" program) |
| seat | S, E1, AY, T |
| second | S, EH1, EH3, K, UH1, N, D |
| secret | S, E1, K, R, I3, T |
| section | S, EH1, EH3, K, SH, UH3, N |
| security | S, EH1, EH3, K, Y, ER, I1, T, Y |
| see | (use "C" program) |
| seize | S, E1, Y, Z |
| select | S, UH1, L, EH1, EH2, K, T |
| sell | S, EH1, EH3, L |
| semi | S, EH1, M, AH1, Y |
| semicolon | S, EH1, M, AH1, Y, K, OO1, O1, L, I2, N |
| send | S, EH1, EH3, N, D |
| sent | (use "cent" program) |
| sentence | S, EH1, N, T, I2, N, DT, S |
| separate | S, EH1, EH3, P, UH1, R, A1, AY, T |
| separate-2 | S, EH1, EH3, P, R, I2, T |
| september | S, EH1, EH3, P, T, EH1, EH3, M, B, ER |
| sequence | S, E1, K, W, EH1, EH3, N, S |
| serial | S, I1, R, Y, UH3, L |
| series | S, I1, R, Y, Z |
| service | S, ER, V, I1, S |
| set | S, EH1, EH3, T |

| Word | Phonetic Program |
|---|---|
| seven | S, EH1, EH3, V, I2, N |
| seventh | S, EH1, EH3, V, I2, N, DT, TH |
| seventy | S, EH1, V, I2, N, D, Y |
| several | S, EH1, V, ER, UH3, L |
| sew | (use "so" program) |
| share | SH, EH3, EH3, ER |
| sharp | SH, AH1, R, P |
| shift | SH, I1, I3, F, T |
| ship | SH, I1, I3, P |
| shop | SH, AH1, UH3, P |
| short | SH, O2, O2, R, T |
| should | SH, IU, IU, IU, D |
| shunt | SH, UH1, UH2, N, T |
| shut | SH, UH1, UH2, T |
| side | S, AH1, EH3, Y, D |
| sierra | S, E1, I3, EH1, R, UH1 |
| signal | S, I1, I3, G, N, UH3, L |
| silver | S, I1, I3, L, V, ER |
| single | S, I1, I3, NG, G, UH3, L |
| six | S, I1, I3, K, PAO, S |
| sixth | S, I1, I3, K, PAO, S, TH |
| sixty | S, I1, I3, K, PAO, T, Y |
| size | S, AH1, EH3, Y, Z |
| skin | S, K, I1, I3, N |
| sky | S, K, AH1, EH3, I3, Y |
| slang | S, L, AE1, EH3, NG |
| slash | S, L, AE1, EH3, SH |
| slave | S, L, A1, AY, Y, V |
| slip | S, L, I1, I3, P |
| slow | S, L, O1, U1 |
| small | S, M, AW, L |
| smell | S, M, EH1, EH3, L |
| smile | S, M, AH1, EH3, I3, UH3, L |
| smoke | S, M, O1, U1, K |
| snow | S, N, OO1, O2, U1 |
| so | S, OO1, O2, U1 |
| soft | S, AW, F, T |
| sold | S, O2, O2, L, L, D |
| solid | S, AH1, UH3, L, I1, D |
| son | (use "sun" program) |
| some | (use "sum" program) |
| sorry | S, AW, R, Y |
| sort | S, O2, O2, R, T |
| sound | S, AH1, UH3, W, N, D |
| source | S, O1, O2, R, S |
| south | S, AH1, UH3, U1, TH |
| space | S, P, A1, AY, Y, S |
| spark | S, P, AH1, R, K |
| speak | S, P, E1, AY, K |
| special | S, P, EH1, EH3, SH, UH3, L |
| speed | S, P, E1, Y, D |
| speech | S, P, E1, Y, T, CH |
| spell | S, P, EH1, EH3, L |
| spend | S, P, EH1, EH3, N, D |
| split | S, P, L, I1, I3, T |
| spoon | S, P, U1, U1, N |
| spring | S, P, R, I1, I3, NG |
| square | S, K, W, EH1, R |
| stack | S, T, AE1, EH3, K |
| stair | (use "stare" program) |
| stand | S, T, AE1, EH3, N, D |
| standard | S, T, AE1, EH3, N, D, ER, D |
| star | S, T, AH1, UH3, R |
| stare | S, T, EH3, EH3, ER |
| start | S, T, AH1, R, T |
| state | S, T, A1, AY, Y, T |
| station | S, T, A1, Y, SH, UH3, N |
| status | S, T, AE1, EH3, T, I2, S |
| steal | (use "steel" program) |
| steel | S, T, E1, Y, L |
| step | S, T, EH1, EH3, P |
| stick | S, T, I1, I3, K |
| stock | S, T, AH1, UH3, K |
| stop | S, T, AH1, UH3, P |
| store | S, T, O2, O2, R |
| strait | (use "straight" program) |
| straight | S, T, R, A1, AY, Y, T |
| street | S, T, R, E1, Y, T |
| stress | S, T, R, EH1, EH3, S |
| string | S, T, R, I1, I3, NG |
| structure | S, T, R, UH1, K, T, CH, ER |
| style | S, T, AH1, EH3, AY, UH3, L |
| subject | S, UH1, UH2, B, D, J, EH1, EH3, K, T |
| substitute | S, UH1, UH3, B, S, T, I3, T, IU, U1, T |
| subtract | S, UH1, UH2, B, T, R, AE1, EH3, K, T |
| sufficient | S, UH1, F, I1, SH, EH3, N, T |
| suggest | S, UH1, UH2, G, D, J, EH1, EH3, S, T |
| suit | S, IU, U1, T |
| suite | S, W, AY, Y, T |
| sum | S, UH1, UH2, M |
| summary | S, UH2, UH2, M, ER, Y |
| summer | S, UH1, UH2, M, ER |
| sun | S, UH1, UH2, N |
| sunday | S, UH1, UH2, N, D, A1, I3, Y |
| super | S, IU, U1, P, ER |
| supply | S, UH2, P, L, AH1, Y |

| Word | Phonetic Program |
|---|---|
| surface | S, ER, F, I2, S |
| surge | S, ER, R, D, J |
| surgery | S, ER, D, J, ER, Y |
| surgical | S, ER, D, J, UH3, K, UH3, L |
| surplus | S, ER, P, L, UH1, S |
| suspend | S, UH1, S, P, EH1, EH3, N, D |
| sweep | S, W, E1, Y, P |
| sweet | (use "suite" program) |
| switch | S, W, I1, I3, T, CH |
| syntax | S, I1, N, T, AE1, EH3, K, PAO, S |
| system | S, I1, S, T, UH3, M |
| T | T, E1, AY, Y |
| table | T, A1, Y, B, UH3, L |
| tail | (use "tale" program) |
| tale | T, A1, Y, UH3, L |
| talk | T, AW, K |
| tangent | T, AE1, EH3, N, D, J, EH3, N, T |
| target | T, AH1, UH3, R, G, I2, T |
| tea | (use "T" program |
| team | T, E1, Y, M |
| technical | T, EH1, EH3, K, N, I3, K, UH3, L |
| tee | (use "T" program) |
| temperature | T, EH1, EH3, M, P, ER, UH1, T, CH, ER |
| ten | T, EH1, EH3, N |
| terminal | T, ER, M, EH3, N, UH2, L |
| test | T, EH1, EH3, S, T |
| than | THV, EH1, EH3, N |
| the | THV, UH1, UH3 |
| then | (use "than" program) |
| theory | TH, AY, I2, R, Y |
| thin | TH, I1, I3, N |
| thing | TH, I1, I3, NG |
| think | TH, I1, I3, NG, K |
| third | TH, ER, R, D |
| thirteen | TH, ER, T, T, E1, Y, N |
| thirty | TH, ER, R, D, Y |
| thousand | TH, AH1, UH3, U1, Z, EH3, N, D |
| three | TH, R, E1, Y |
| threw | (use "through" program) |
| through | TH, R, IU, U1 |
| thursday | TH, ER, R, Z, D, A1, I3, Y |
| ticket | T, I1, I3, K, EH3, T |
| till | T, I1, I3, L |
| time | T, AH1, EH3, Y, M |
| tire | T, AH1, EH3, AY, R |
| title | T, UH3, AH2, Y, T, UH3, L |
| to | (use "two" program) |
| today | T, U1, D, A1, I3, Y |
| toilet | T, O1, EH3, I3, L, I3, T |
| toil | T, O2, O2, OO1, L |
| tomorrow | T, U1, M, AH1, R, O1, U1 |
| ton | T, UH1, UH2, N, N |
| tone | T, O1, U1, N |
| too | (use "two" program) |
| tool | T, U1, U1, L |
| total | T, O1, U1, T, UH3, L |
| touch | T, UH1, UH3, T, CH |
| towel | T, AH1, W, UH3, L |
| trace | T, R, A1, AY, Y, S |
| trade | T, R, A1, AY, Y, D |
| train | T, R, A1, AY, Y, N |
| transact | T, R, AE1, EH3, N, S, AE1, EH3, K, T |
| transfer | T, R, AE1, EH3, N, S, F, ER |
| transistor | T, R, AE1, N, Z, I1, S, T, ER |
| transmit | T, R, AE1, EH3, N, Z, M, I1, I3, T |
| transport | T, R, AE1, EH3, N, S, P, O2, O2, R, T |
| transportation | T, R, AE1, N, S, P, ER, T, A1, AY, SH, UH3, N |
| travel | T, R, AE1, EH3, V, UH3, L |
| triangle | T, R, AH1, I3, AE1, EH3, NG, G, UH3, L |
| trouble | T, R, UH3, UH1, B, UH3, L |
| truck | T, R, UH1, UH2, K |
| true | T, R, IU, U1, U1 |
| trust | T, R, UH1, UH2, S, T |
| try | T, R, AH1, EH3, I3, Y |
| tuesday | T, IU, U1, U1, Z, D, A1, Y |
| tune | T, IU, U1, U1, N |
| turn | T, ER, R, N |
| twelve | T, W, EH1, EH3, UH3, L, V |
| twenty | T, W, EH1, EH3, N, T, Y |
| two | T, IU, U1, U1 |
| type | T, UH3, AH2, Y, P |
| U | Y1, IU, U1, U1 |
| ultra | UH3, UH2, L, T, R, UH1 |
| under | UH2, UH2, N, D, ER |
| uniform | Y1, IU, U1, N, I3, F, O1, R, M |
| until | UH2, UH2, N, T, I1, I3, L |
| up | UH1, UH2, P |
| urgent | R, R, D, J, I3, N, T |
| us | UH1, UH2, S |
| use | Y1, IU, U1, U1, Z |
| use-2 | Y1, IU, U1, S |
| V | V, E1, AY, Y |

| Word | Phonetic Program |
|---|---|
| vacant | V, A1, Y, K, EH3, N, T |
| valid | V, AE1, UH3, L, I1, D |
| vary | (use "very" program) |
| value | V, AE1, EH3, L, Y1, IU, U1 |
| vendor | V, EH1, EH3, N, D, ER |
| vent | V, EH1, EH3, N, T |
| verify | V, EH1, R, I3, F, AH1, EH3, Y |
| very | V, EH1, R, Y |
| via | V, E1, AY, UH2, UH3 |
| victor | V, I1, I3, K, T, ER |
| voice | V, O1, UH3, I3, AY, S |
| void | V, O1, UH3, I3, AY, D |
| volt | V, O2, O2, L, T |
| volume | V, AH1, UH3, L, Y1, IU, U1, M |
| W | D, UH1, B, UH3, L, Y1, IU, U1 |
| wage | W, A1, AY, Y, D, J |
| wait | W, A1, AY, Y, T |
| want | W, AH1, UH3, N, T |
| was | W, UH1, UH3, Z |
| wash | W, AW, SH |
| water | W, AH1, UH3, T, ER |
| watt | W, AH1, UH3, T |
| wave | W, A1, AY, Y, V |
| way | (use "weigh" program) |
| we | W, E1, Y |
| weak | (use "week" program) |
| weapon | W, EH2, EH2, P, UH1, N |
| wear | (use "where" program) |
| wednesday | W, EH1, N, Z, D, A1, I3, Y |
| week | W, E1, Y, K |
| weigh | W, A2, A2, Y |
| weight | (use "wait" program) |
| went | W, EH1, EH3, N, T |
| west | W, EH1, EH3, S, T |
| wet | W, EH1, EH3, T |
| what | W, UH3, UH1, T |
| wheel | W, E1, Y, L |
| when | W, EH1, EH3, N |
| where | W, EH3, A2, EH3, R |
| which | W, I1, I3, T, CH |
| while | W, AH1, EH3, I1, UH3, L |
| whiskey | W, I1, I3, S, K, AY, Y |
| white | W, UH3, AH2, Y, T |
| who | H, IU, U1, U1 |
| whole | (use "hole" program) |
| why | (use "Y" program) |
| will | W, I1, I3, L |
| window | W, I1, N, D, O1, U1 |
| winter | W, I1, I3, N, T, ER |
| wire | W, AH1, EH3, AY, R |
| with | W, I1, I3, TH |
| withdraw | W, I1, I3, TH, D, R, AW |
| without | W, I1, I3, TH, UH2, AH2, U1, T |
| won | (use "one" program) |
| word | W, ER, R, D |
| work | W, ER, R, K |
| write | (use "right" program) |
| wrong | R, AW, NG |
| X | EH1, EH2, K, PAO, S |
| x-ray | EH1, EH2, K, PAO, S, R, A1, I3. Y |
| Y | W, AH1, EH3, I3, Y |
| yankee | Y1, AE1, EH3, NG, K, E1, Y |
| yard | Y1, AH1, R, D |
| year | Y1, AY, I3, R |
| yellow | Y1, EH1, EH3, L, O1, U1 |
| yes | Y1, EH3, EH1, S |
| yesterday | Y1, EH3, EH1, S, T, ER, D, A1, I3, Y |
| yet | Y1, EH1, EH3, T |
| you | (use "U" program) |
| your | Y, O2. O2, R |
| you're | (use "your" program) |
| Z | Z, E1, Y |
| zap | Z, AE1, EH3, P |
| zero | Z, AY, I1, R, O1, U1 |
| zone | Z, O1, U1, N |
| zulu | Z, IU, U1, L, IU, U1 |

| Prefixes | |
|---|---|
| con... | K, UH1, N |
| dis... | D, I1, S |
| en... | EH1, N |
| in... | I1, N |
| non... | N, AH1, UH3, N |
| pre... | P, R, E1 |
| re... | R, E1 |
| un... | UH1, N |

| Suffixes | |
|---|---|
| ...d | D |
| ...ed | I2, D |
| ...er | ER |
| ...es | I2, Z |
| ...ful | F, UH3, L |
| ...ing | I2, NG |

| Word | Phonetic Program |
|---|---|
| ...less | L, EH2, S |
| ...ly | L, Y |
| ...ment | M, EH3, N, T |
| ...ness | N, EH3, S |
| ...s | S |
| ...t (...ed) | T |
| ...tion (...sion) | SH, UH3, N |
| ...teen | T, E1, Y, N |
| ...ward | W, ER, D |
| ...y | Y |
| ...z (...es) | Z |

# INDEX

# The SYBEX Library

*INTRODUCTION TO COMPUTERS*

### DON'T (or How to Care for Your Computer)

**by Rodnay Zaks** 214 pp., 100 illustr., Ref. 0-065
The correct way to handle and care for all elements of a computer system, including what to do when something doesn't work.

### YOUR FIRST COMPUTER

**by Rodnay Zaks** 258 pp., 150 illustr., Ref. 0-045
The most popular introduction to small computers and their peripherals: what they do and how to buy one.

### INTERNATIONAL MICROCOMPUTER DICTIONARY

120 pp., Ref. 0-067
All the definitions and acronyms of microcomputer jargon defined in a handy pocket-size edition. Includes translations of the most popular terms into ten languages.

### FROM CHIPS TO SYSTEMS: AN INTRODUCTION TO MICROPROCESSORS

**by Rodnay Zaks** 552 pp., 400 illustr., Ref. 0-063
A simple and comprehensive introduction to microprocessors from both a hardware and software standpoint: what they are, how they operate, how to assemble them into a complete system.

*FOR YOUR COMMODORE 64/VIC-20*

### THE COMMODORE 64™/VIC-20™ BASIC HANDBOOK

**by Douglas Hergert,** 144 pp., Ref. 0-116
A complete listing with descriptions and instructive examples of each of the Commodore 64 BASIC keywords and functions. A handy reference guide, organized like a dictionary.

### THE EASY GUIDE TO YOUR COMMODORE 64™

**by Joseph Kascmer** 160 pp., illustr., Ref. 0-0126
A friendly introduction to using the Commodore 64.

### YOUR FIRST VIC-20™ PROGRAM

**by Rodnay Zaks** 150 pp. illustr., Ref. 0-129
A fully illustrated, easy-to-use, introduction to VIC-20 BASIC programming. Will have the reader programming in a matter of hours.

*BASIC*

### YOUR FIRST BASIC PROGRAM

**by Rodnay Zaks** 150pp. illustr. in color, Ref. 0-129
A "how-to-program" book for the first time computer user, aged 8 to 88.

## FIFTY BASIC EXERCISES

**by J. P. Lamoitier** 232 pp., 90 illustr., Ref. 0-056

Teaches BASIC by actual practice, using graduated exercises drawn from everyday applications. All programs written in Microsoft BASIC.

## INSIDE BASIC GAMES

**by Richard Mateosian** 348 pp., 120 illustr., Ref. 0-055

Teaches interactive BASIC programming through games. Games are written in Microsoft BASIC and can run on the TRS-80, Apple II and PET/CBM.

## BASIC FOR BUSINESS

**by Douglas Hergert** 224 pp., 15 illustr., Ref. 0-080

A logically organized, no-nonsense introduction to BASIC programming for business applications. Includes many fully-explained accounting programs, and shows you how to write them.

## EXECUTIVE PLANNING WITH BASIC

**by X. T. Bui** 196 pp., 19 illustr., Ref. 0-083

An important collection of business management decision models in BASIC, including Inventory Management (EOQ), Critical Path Analysis and PERT, Financial Ratio Analysis, Portfolio Management, and much more.

## BASIC PROGRAMS FOR SCIENTISTS AND ENGINEERS

**by Alan R. Miller** 318 pp., 120 illustr., Ref. 0-073

This book from the "Programs for Scientists and Engineers" series provides a library of problem-solving programs while developing proficiency in BASIC.

# *ASSEMBLY LANGUAGE PROGRAMMING*

## PROGRAMMING THE 6502

**by Rodnay Zaks** 386 pp., 160 illustr., Ref. 0-046

Assembly language programming for the 6502, from basic concepts to advanced data structures.

## 6502 APPLICATIONS

**by Rodnay Zaks** 278 pp., 200 illustr., Ref. 0-015

Real-life application techniques: the input/output book for the 6502.

## ADVANCED 6502 PROGRAMMING

**by Rodnay Zaks** 292 pp., 140 illustr., Ref. 0-089

Third in the 6502 series. Teaches more advanced programming techniques, using games as a framework for learning.

## PROGRAMMING THE Z80

**by Rodnay Zaks** 624 pp., 200 illustr., Ref. 0-069

A complete course in programming the Z80 microprocessor and a thorough introduction to assembly language.

## Z80 APPLICATIONS

**by James W. Coffron** 288 pp., illustr., Ref. 0-094

Covers techniques and applications for using peripheral devices with a Z80 based system.

Now you can do more with your VIC-20 than play games!

# THE VIC-20 CONNECTION

**The VIC-20 Connection** will show you how easy it is to use your computer together with household devices. You will quickly learn the simple techniques for using your VIC-20 to control:

- a home security system
- a home temperature system
- a voice synthesizer to make your computer talk
- and more home control appliances

The VIC-20 computer is well suited for connecting to non-computer devices. With this book and a little imagination, there is no limit to the exciting and useful ways to use your VIC-20 computer.

## About the author:

James W. Coffron is a computer systems engineer specializing in the design and testing of microprocessor-based systems. He has taught seminars in both academic and industrial settings, and is the author of several books, including the **Apple Connection**, and **Z80 Applications.** He holds an MSEE degree from Santa Clara University.

ISBN 0-89588-128-4